Infrastructure de gestion de la confiance sur internet

Van-Hoan Vu

Ecole Nationale Supérieure des Mines
SAINT-ETIENNE

NNT : 2010 EMSE 0585

THÈSE

présenté par

Hoan VU

pour obtenir le grade de

Docteur de l'École Nationale Supérieure des Mines de Saint-Etienne

spécialité INFORMATIQUE

INFRASTRUCTURE DE GESTION DE LA CONFIANCE SUR INTERNET

soutenue à Saint-Étienne le 3 décembre 2010

Membres de jury

Président :	Pierre MARET	Professeur, Université Jean Monnet
Rapporteurs :	Robert MAHL	Professeur, École des Mines de Paris
	Patrick BELLOT	Professeur, Télécom ParisTech
Examinateurs :	Mohand-Saîd HACID	Professeur, Université Claude Bernard
	Pierre MARET	Professeur, Université Jean Monnet
Directeurs :	Jean-Jacques GIRARDOT	Maître de recherche, ENSM Saint-Étienne
	Philippe JAILLON	Ingénieur de recherche, ENSM Saint-Étienne

Spécialités doctorales :	**Responsables :**
SCIENCES ET GENIE DES MATERIAUX	J. DRIVER Directeur de recherche – Centre SMS
MECANIQUE ET INGENIERIE	A. VAUTRIN Professeur – Centre SMS
GENIE DES PROCEDES	G. THOMAS Professeur – Centre SPIN
SCIENCES DE LA TERRE	B. GUY Maître de recherche – Centre SPIN
SCIENCES ET GENIE DE L'ENVIRONNEMENT	J. BOURGOIS Professeur – Centre SITE
MATHEMATIQUES APPLIQUEES	E. TOUBOUL Ingénieur – Centre G2I
INFORMATIQUE	O. BOISSIER Professeur – Centre G2I
IMAGE, VISION, SIGNAL	JC. PINOLI Professeur – Centre CIS
GENIE INDUSTRIEL	P. BURLAT Professeur – Centre G2I
MICROELECTRONIQUE	Ph. COLLOT Professeur – Centre CMP

Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat (titulaires d'un doctorat d'État ou d'une HDR)

AVRIL	Stéphane	MA	Mécanique & Ingénierie	CIS
BATTON-HUBERT	Mireille	MA	Sciences & Génie de l'Environnement	SITE
BENABEN	Patrick	PR 1	Sciences & Génie des Matériaux	CMP
BERNACHE-ASSOLLANT	Didier	PR 0	Génie des Procédés	CIS
BIGOT	Jean-Pierre	MR	Génie des Procédés	SPIN
BILAL	Essaïd	DR	Sciences de la Terre	SPIN
BOISSIER	Olivier	PR 1	Informatique	G2I
BORBELY	Andras	MR	Sciences et Génie des Matériaux	SMS
BOUCHER	Xavier	MA	Génie Industriel	G2I
BOUDAREL	Marie-Reine	PR 2	Génie Industriel	DF
BOURGOIS	Jacques	PR 0	Sciences & Génie de l'Environnement	SITE
BRODHAG	Christian	DR	Sciences & Génie de l'Environnement	SITE
BURLAT	Patrick	PR 2	Génie industriel	G2I
COLLOT	Philippe	PR 1	Microélectronique	CMP
COURNIL	Michel	PR 0	Génie des Procédés	SPIN
DAUZERE-PERES	Stéphane	PR 1	Génie industriel	CMP
DARRIEULAT	Michel	IGM	Sciences & Génie des Matériaux	SMS
DECHOMETS	Roland	PR 1	Sciences & Génie de l'Environnement	SITE
DESRAYAUD	Christophe	MA	Mécanique & Ingénierie	SMS
DELAFOSSE	David	PR 1	Sciences & Génie des Matériaux	SMS
DOLGUI	Alexandre	PR 1	Génie Industriel	G2I
DRAPIER	Sylvain	PR 2	Mécanique & Ingénierie	SMS
DRIVER	Julian	DR 0	Sciences & Génie des Matériaux	SMS
FEILLET	Dominique	PR 2	Génie Industriel	CMP
FOREST	Bernard	PR 1	Sciences & Génie des Matériaux	CIS
FORMISYN	Pascal	PR 1	Sciences & Génie de l'Environnement	SITE
FORTUNIER	Roland	PR 1	Sciences & Génie des Matériaux	SMS
FRACZKIEWICZ	Anna	DR	Sciences & Génie des Matériaux	SMS
GARCIA	Daniel	MR	Génie des Procédés	SPIN
GIRARDOT	Jean-Jacques	MR	Informatique	G2I
GOEURIOT	Dominique	MR	Sciences & Génie des Matériaux	SMS
GRAILLOT	Didier	DR	Sciences & Génie de l'Environnement	SITE
GROSSEAU	Philippe	MR	Génie des Procédés	SPIN
GRUY	Frédéric	MR	Génie des Procédés	SPIN
GUY	Bernard	MR	Sciences de la Terre	SPIN
GUYONNET	René	DR	Génie des Procédés	SPIN
HERRI	Jean-Michel	PR 2	Génie des Procédés	SPIN
INAL	Karim	PR 2	Microélectronique	CMP
KLÖCKER	Helmut	DR	Sciences & Génie des Matériaux	SMS
LAFOREST	Valérie	CR	Sciences & Génie de l'Environnement	SITE
LERICHE	Rodolphe	CR CNRS	Mécanique et Ingénierie	SMS
LI	Jean-Michel	EC (CCI MP)	Microélectronique	CMP
LONDICHE	Henry	MR	Sciences & Génie de l'Environnement	SITE
MALLIARAS	George Grégory	PR 1	Microélectronique	CMP
MOLIMARD	Jérôme	MA	Mécanique et Ingénierie	SMS
MONTHEILLET	Frank	DR 1 CNRS	Sciences & Génie des Matériaux	SMS
PERIER-CAMBY	Laurent	PR 2	Génie des Procédés	SPIN
PIJOLAT	Christophe	PR 1	Génie des Procédés	SPIN
PIJOLAT	Michèle	PR 1	Génie des Procédés	SPIN
PINOLI	Jean-Charles	PR 0	Image, Vision, Signal	CIS
STOLARZ	Jacques	CR	Sciences & Génie des Matériaux	SMS
SZAFNICKI	Konrad	MR	Sciences & Génie de l'Environnement	SITE
THOMAS	Gérard	PR 0	Génie des Procédés	SPIN
TRIA	Assia		Microélectronique	CMP
VALDIVIESO	François	MA	Sciences & Génie des Matériaux	SMS
VAUTRIN	Alain	PR 0	Mécanique & Ingénierie	SMS
VIRICELLE	Jean-Paul	MR	Génie des procédés	SPIN
WOLSKI	Krzysztof	DR	Sciences & Génie des Matériaux	SMS
XIE	Xiaolan	PR 1	Génie industriel	CIS

Glossaire :

PR 0	Professeur classe exceptionnelle
PR 1	Professeur 1ère classe
PR 2	Professeur 2ème classe
MA(MDC)	Maître assistant
DR	Directeur de recherche
Ing.	Ingénieur
MR(DR2)	Maître de recherche
CR	Chargé de recherche
EC	Enseignant-chercheur
IGM	Ingénieur général des mines

Centres :

SMS	Sciences des Matériaux et des Structures
SPIN	Sciences des Processus Industriels et Naturels
SITE	Sciences Information et Technologies pour l'Environnement
G2I	Génie Industriel et Informatique
CMP	Centre de Microélectronique de Provence
CIS	Centre Ingénierie et Santé

Remerciements

Je tiens à exprimer tout d'abord mes remerciements aux membres de jury, qui ont accepté d'évaluer mon travail de thèse.

Je voudrais remercier M. Pierre Maret d'avoir présidé le jury cette thèse. Je tiens à remercier M. Robert Mahl et M. Patrick Bellot d'avoir accepté d'être les rapporteurs de ce manuscrit. Leur remarques et suggestions pertinentes lors de la lecture de mon rapport m'ont permis d'amiliorer ce mémoire. Je remercie M. Mohand-Saîd Hacid pour avoir accepté d'examiner ce manuscrit et de faire partie de mon jury de thèse.

J'exprime toute ma gratitude à M. Jean-Jacques Girardot et M. Philippe Jaillon, mes directeurs de thèse, pour leur encadrement de qualité, leur patience et gentillesse durant les longues années de ma thèse. Ils m'ont beaucoup aidé à orienter ce travail, et également m'ont encouragé pendant les périodes difficiles.

Je tiens à remercier Marc Roelens et Xavier Serpaggi pour les discussions et conseils lors du travail réalisé ensemble dans l'équipe, ainsi que leur relecture de mon manuscrit.

Je remercie tous les membres du centre G2I pour leur soutien et encouragement, Olivier, Laurent, Michel, Philippe B., Roland, Gauthier P., Annie, Amélie, Olga, Ali, Reda, Steve, Grégory D., Grégory S., Antoine, Alain M, Nilou, Jean-François. J'adresse un remerciement tout particulier à Marie-Line pour tous les supports de la procédure administrative et son aide dans la correction de mon manuscrit, à Ali pour son support lors de la préparation de ma soutenance.

Un grand merci aussi à mes amis vietnamiens à Saint-Étienne qui m'ont apporté leur soutien moral et leur amitié pendant le temps que nous avons passé ensemble. En particulier à Thao, Duyen, Thu, Cong, Thuy, Hien qui m'ont aidé à la préparation de la célébration de ma soutenance.

Finalement, j'adresse mes profonds remerciements à toute ma famille qui a toujours été présente à mon côté au long de la thèse.

Table des matières

Table des figures

Chapitre 1

Introduction

Le problème de la confiance est une question que nous avons à nous poser tous les jours, dans la vie quotidienne, dans les sciences sociales, dans l'économie, le commerce ou dans le domaine des applications Internet. La notion de confiance représente les relations entre deux acteurs avec quelques propriétés spécifiques. Nous allons présenter des problèmes liés à la confiance dans quelques domaines différents de manière à préciser le contexte de notre étude.

Dans la vie quotidienne, la confiance est un facteur important dans l'établissement de relations entre personnes. Plusieurs questions concernant la confiance se posent : comment pouvons-nous faire confiance à une personne pour mener à bien une opération, une transaction avec elle, notamment s'il s'agit de quelqu'un que nous n'avons jamais rencontré ? Lorsqu'un recruteur veut pourvoir un poste dans son entreprise, son objectif est de trouver le candidat qui correspond le mieux au poste. Comment peut-il faire pour avoir confiance dans le CV des candidats lors de la phase de sélection ? Pour ce faire, le recruteur évaluera sans doute le candidat par une méthode classique : il vérifiera son parcours, ses expériences et peut-être même demandera-t-il des informations complémentaires aux personnes citées comme références. À partir de cette évaluation, le recruteur aura pu évaluer la confiance qu'il a dans le profil du candidat et décider ou non de son embauche.

Dans le monde économique et le commerce, les problèmes de confiance sont très souvent la clé des transactions commerciales. Supposons qu'une société A, *en France*, veuille importer les produits d'une société B, *en Chine*. La société A se pose des questions concernant la confiance qu'elle a dans la société B : est ce que les produits de B respectent des critères de qualité de l'EU ? Est-ce que A recevra les produits après avoir payé la marchandise commandée ? Bien évidement, la société B se pose, elle aussi, des questions sur A : est-ce que A est un bon client ? Est-ce qu'elle sera payée après avoir livré la commande de A ?

Dans le domaine des applications Internet, le problème de la confiance devient encore plus important. De nos jours, l'usage d'Internet se développe très rapidement et couvre de plus en plus de domaines d'applications, telles que les applications bancaires en ligne, les systèmes de e-commerce, les communautés virtuelles... La question de la *confiance* se pose pour l'application elle même (le système), mais aussi pour les participants impliqués, les utilisateurs du système. Que représente la confiance dans ce domaine ? Comment pouvons-nous prendre une décision sur la confiance que nous avons dans la réalisation d'une action ? Nous présenterons dans la section suivante ce problème au travers de quelques scénarios d'applications.

1.1 Confiance pour les applications Internet

Nous présentons ici quelques scénarios d'applications pour lesquels l'évaluation de la confiance est un problème reconnu. Il s'agit des problèmes de confiance dans les sources d'informations, de la confiance dans l'identité d'une entité virtuelle et des problèmes de confiance dans le commerce électronique. Les problèmes liés aux risques et à la négociation en vue d'établir la confiance sont aussi les questions importantes de ce sujet d'étude.

Sources d'information

Imaginons un scénario de recherche d'information sur l'Internet concernant un événement. Nous pouvons trouver plusieurs documents à partir de sources d'informations diverses. Les questions de confiance sur les documents retrouvés se posent de la manière suivante : « est-ce que nous pouvons faire confiance à l'information donnée dans le document ? » Pour répondre à cette question, nous pourrions vérifier que le document est certifié par un tiers sûr, ou bien que le document est fourni par une source de confiance. Nous avons besoin d'une solution de confiance pour évaluer une telle source d'information.

Par extension, nous pourrions vouloir évaluer les services d'un fournisseur en ligne. Nous voudrions savoir si ses services sont de qualité pour décider de les utiliser ou pas. La question de la confiance est encore une fois posée. Dans le cas concret d'un site de formation, nous voudrions savoir si ses cours présentés sont intéressants, s'ils ont vu leurs contenus évalués par des formateurs compétents, s'ils sont accompagnés d'un support en ligne (i.e. un intervenant disponible 24h/24 et 7j/7 pour répondre aux questions) ? L'utilisation d'une infrastructure de confiance utilisant un modèle de réputation est peut-être une bonne solution pour ce scénario. Elle fournirait aux utilisateurs des éléments leur permettant d'évaluer la confiance dans le fournisseur de ce service.

Identité d'un acteur virtuel

Dans le monde de l'Internet, nous travaillons dans un monde virtuel où les personnes réelles sont dématérialisées sous la forme d'acteurs virtuels. En conséquence, une question qui se pose souvent est : « est-ce que nous pouvons faire confiance à cette entité pour entreprendre une action ? ».

Chaque acteur virtuel est désigné par une identité, et il est difficile de vérifier que cette identité représente bien la personne avec qui nous voulons entreprendre l'action. De plus, disposer d'une identité vérifiable n'est pas une condition suffisante, un *pirate* dispose lui aussi d'une véritable identité pour mener à bien ses attaques. Une solution à ce problème d'identité est de disposer d'un système qui permette d'établir un niveau de confiance suffisant avec des individus inconnus en plus des mécanismes d'authentification standards à base de certificats.

Étudions encore quelques exemples concrets : pour accéder au site des impôts (ministère des finances français), nous avons besoin d'un certificat qui est associé à l'identité fiscale d'une personne donnée. L'accès à des sites sécurisés se fait en général avec des certificats où l'identité du porteur est attestée par un tiers sûr (un tiers de confiance). Un autre exemple typique est l'identification de l'émetteur d'un email, car le protocole ne nous permet pas de vérifier qui nous l'a envoyé. L'utilisation de PGP [75, 76] est, par exemple, une solution pour résoudre ce problème.

Commerce électronique

Le commerce électronique est un cas d'usage typique où la question de la confiance se pose pour décider de mener à bien ou non une transaction électronique entre un client et un service marchand en ligne. Il est à noter que cette transaction se déroule en l'absence de tout contact physique et très souvent sans que les acteurs ne se soient rencontrés au préalable.

- Dans quelles conditions le client peut-il faire confiance à un site pour sa commande ? Il ne sait pas s'il recevra le produit commandé après l'avoir payé ; un site malhonnête peut très bien ne pas envoyer ce produit après en avoir reçu le paiement, ou bien envoyer un produit qui n'est pas conforme à ce qui a été commandé.
- Le site de commerce en ligne peut-il faire confiance au client pour valider la commande ? Il n'est pas sûr d'être payé par le client une fois le produit expédié. Mais aussi, un client malhonnête peut utiliser une carte bancaire frauduleuse pour en effectuer le paiement.

Ces exemples nous montrent qu'un mécanisme de confiance est indispensable pour assurer le développement et le fonctionnement de tels systèmes. La solution à ce problème est la définition d'une infrastructure de confiance dont l'objet est d'aider à la prise de décision en ce qui concerne la confiance dans l'autre partie. Ce mécanisme de décision s'appuie sur la spécification d'une politique de confiance et sur des mécanismes de négociation en vue d'établir la confiance entre les différents acteurs du système.

Problème du risque

La prise en compte du risque est un point important pour tous les mécanismes de confiance. Dans le contexte réel, lorsque nous devons prendre une décision, par exemple réaliser un investissement financier, entreprendre une coopération... nous tentons toujours d'estimer les risques que nous allons prendre au préalable. Le risque est en rapport direct avec la sécurité. Dans le cas où la sécurité est assurée, il n'y a donc pas de risque et la question de la confiance ne devrait pas se poser. Par contre, quand il existe un risque, on se trouve dans un contexte incertain et la question sur la confiance se pose. Dans cette situation, on a besoin d'un mécanisme d'aide à la décision pour évaluer les conséquences possibles de l'action à entreprendre et leurs différentes probabilités d'occurrence ; ces éléments servant à évaluer les risques et par conséquent à nous permettre de faire ou ne pas faire confiance à notre correspondant.

Problèmes de négociation de la confiance

Le but de la négociation de la confiance est de permette aux acteurs d'une transaction de négocier pour obtenir un niveau de confiance qui les satisfait simultanément. Cette négociation peut se traduire par des échanges de données tels que des certificats, des historiques d'interaction ou même tout ou partie des règles de leurs politiques de confiance. Nous allons présenter dans le paragraphe suivant une situation typique de négociation de la confiance : la négociation entre un client et un marchand ayant pour objectif de mener à bien une transaction commerciale.

Dans une communauté constituée de plusieurs entités, deux de ces entités veulent établir un contact ayant un niveau de confiance suffisant avec pour objectif de réaliser une transaction. Chacune des entités se définit une politique de confiance pour protéger ses ressources, cette politique peut tirer profit d'informations sur l'historique des

transactions qui ont été menées avec les autres membres de la communauté.

Pour résoudre les problèmes de ce type, nous avons besoin d'une infrastructure de confiance. Cette infrastructure se doit d'avoir les composants suivants :

- un modèle de la confiance ;
- un modèle de risque ;
- un formalisme pour exprimer sa politique de confiance ;
- un mécanisme d'évaluation .

1.2 Infrastructure de la confiance

Pour résoudre les problèmes de confiance que nous avons évoqués précédemment, nous avons besoin d'une infrastructure qui permette de passer de la modêlisation du problème au niveau concret et pratique. Cette infrastructure devrait avoir les fonctions suivantes :

- Formalisation des relations de confiance entre les participants au système. Il s'agit d'un modèle de la confiance qui permet de spécifier et de représenter les relations sous une forme calculable ou prouvable. La formalisation choisie indique quels sont les paramètres qui seront pris en compte pour évaluer la confiance.
- Modélisation de la politique.
- Négociation de la confiance. Il s'agit de permettre aux participants de négocier les éléments nécessaires pour établir la confiance indispensable à la réalisation d'une transaction.

1.3 Objectif de la thèse

De nombreuses infrastructures de gestion de la confiance dédiées à des domaines d'applications spécifiques ont déjà été proposées. Quelques unes d'entre elles proposent des modèles où le niveau de confiance est calculé à partir de l'expérience et des recommandations des entités impliquées dans le système. Les résultats sont utilisés pour décider de donner la permission à l'autre partie d'entreprendre les actions demandées. Ce type de solution est généralement utilisé par les systèmes *peer-to-peer*.

Une autre approche qui a été utilisée lors de la conception des premiers systèmes de gestion de la confiance tel que *KeyNote [65] ou Trustbuilder [83], est basée sur l'utilisation de politiques*. Les *principals*[1] du système décrivent les règles de protection de leurs données des politiques qui leur sont propres. Les autorisations sont représentées par un mécanisme de délégation qui utilise des *credentials* (des qualifications, par exemple des certificats signés) fournis par les parties. La vérification de la conformité, pour une action donnée, entre la politique du *principal* et les *credentials* présentés permet de rendre une réponse binaire à l'utilisateur : action autorisée ou refusée.

Dans le contexte d'une application réelle, on peut imaginer que la décision liée à la confiance sera plus sûre si elle est prise en combinant toutes les sources d'informations à notre disposition, des *credentials* et des expériences. Si les conséquences des décisions sont moins importantes, nous pouvons imaginer n'utiliser que les informations relatives à l'expérience ou seulement les *credentials*.

[1]Le terme *principal* fait référence à la personne qui est responsable d'entreprendre une action. Aujourd'hui, en français et sauf dans le cadre de la sécurité, il n'est plus utilisé que pour désigner le *principal* d'un collège. Le terme sera défini au chapitre 2.

Un autre critère très important qu'il nous faut aborder lors de la conception de notre système de gestion de confiance est la décentralisation de ce système. La décentralisation doit être effective au niveau des données, des politiques mais aussi de la prise de décision. Au niveau des données, chaque entité conserve les données qui lui sont propres comme ses expériences et ses credentials. Éventuellement, les données relatives à l'expérience peuvent être partagées avec d'autres entités du système et servir à des mécanismes de recommandation. Chaque entité gérant sa propre politique de sécurité et ses propres décisions, la décentralisation se fait naturellement. Cette décentralisation globale du système est nécessaire pour en réduire la complexité dans des environnements vastes et complexes tel que l'Internet et permet de rendre le système utilisable dans la pratique.

L'objectif de cette thèse est de proposer un modèle de gestion de la confiance décentralisé qui utilisera toutes les sources d'informations à sa disposition pour décider d'accorder ou non sa confiance. De plus, le modèle proposé permettra aux différents participants de négocier, action par action, une confiance satisfaisant leurs différentes politiques.

1.4 Contributions

La contribution principale de cette thèse pour le domaine de la négociation de la confiance est un modèle d'infrastructure de confiance innovant et intégrant de nouveaux aspects par rapport aux systèmes existants. Le modèle proposé et l'infrastructure qui en découle comprennent les composants suivants : un modèle qui formalise la confiance et la prise de décision dans le cadre de l'évaluation de la confiance ; un modèle de négociation dédié à l'établissement de la confiance ; un modèle des risques et son utilisation. Nous avons aussi développé un prototype et l'avons utilisé pour implémenter un scénario simple de négociation dans le cadre d'une application de e-commerce.

Formalisation de la confiance

La notion de la confiance, de notre point de vue, est associée strictement au risque. Nous proposons un modèle de prise de décision lié à la confiance où toutes les informations telles que l'expérience, les qualifications (*credentials*) et le risque sont pris en compte. Ce travail adapte à la négociation le modèle de structure d'événements, proposé par le projet SECURE [15] et le système de réputation de Krukow *et al.* [59].

Négociation de la confiance

En utilisant la formalisation de la confiance que nous venons de présenter ci-dessus, nous proposons un modèle de négociation qui permet d'établir simultanément la confiance entre les deux parties d'une transaction. Le système de négociation est distribué et utilise indépendamment la politique de chaque partie. Ce processus de négociation raisonne sur la politique et vérifie à chaque étape que la politique de confiance peut être satisfaite.

Modèle de risque

Le risque est un facteur important dans la prise de décision. Sa prise en compte repose sur un modèle qui nous permet d'en calculer le niveau. Ce modèle utilise, lui aussi, différentes sources informations telles que les expériences passées, le montant de la transaction, etc.

Prototype d'application

Au niveau de l'implémentation de l'infrastructure, nous apportons les éléments suivants : un scénario d'application qui met en œuvre la négociation dans le cadre d'une transaction en ligne (vente/achat) ; les politiques des différents acteurs exprimées dans un format XML ; un module de vérification de la politique locale de chacune des parties ; un module de calcul du risque et enfin, un module implémentant une stratégie simple pour piloter la négociation. Notre implémentation n'a pas encore complètement abouti sur l'ensemble des points, mais nous disposons d'un prototype partiellement fonctionnel.

1.5 Plan du manuscrit

Ce manuscrit se compose de 9 chapitres. Les deux premiers chapitres (2 et 3) constituent un état de l'art. Les 4 chapitres suivants traitent de notre travail pour résoudre le problème posé. Le détail du travail dans les différents chapitres est organisé comme suit :

Le chapitre 1 est une introduction au problème de la confiance. Nous y précisons également l'objectif et la contribution de la thèse.

Le chapitre 2 présente de manière générale les notions de base de la gestion de confiance. Nous y citons les différentes définitions de la confiance, du risque et de la gestion de la confiance communément admises dans la littérature. Nous y précisons également notre point de vue sur ces notions. Ensuite, nous abordons un ensemble (non exhaustif) d'exemples typiques d'applications qui nous semblent avoir besoin des mécanismes de gestion de confiance et de négociation de la confiance.

Le chapitre 3 expose les principales solutions d'infrastructure de confiance décrites dans la littérature. Nous commençons par une présentation sur les différentes approches des systèmes de confiance : approches basées sur les politiques et approches basées sur l'expérience. Par la suite, nous détaillons quelques solutions significatives telles que KeyNote [65], Trustbuilder [83], SULTAN [35, 28], le projet SECURE [15] ou le modèle présenté par Krukow [59]. Enfin, nous analysons ces solutions, nous en citons leurs points forts et leurs points faibles afin de préparer les bases de notre proposition.

Le chapitre 4 présente le principe général de notre modèle de confiance. Nous définissons les contraintes appliquées à notre modèle et nous le comparons avec les systèmes existants. Ensuite, nous présentons l'infrastructure à base d'événements que nous avons utilisée pour concevoir notre système.

Le chapitre 5 décrit notre formalisation de la confiance et le modèle de prise de décision associé. Nous détaillerons de quelle manière il nous est possible d'utiliser toutes les sources d'informations dans nos politiques de confiance.

Le chapitre 6 présente un modèle de négociation de confiance qui utilise une procédure globale de décision. Ce modèle de négociation de confiance est constitué de trois composants principaux : les politiques et l'historique des transcations, le protocole d'échange entre les deux parties et le processus de vérification de la conformité entre la politique et l'action. Ce modèle est illustré par un scénario d'application de négociation.

Le chapitre 7 propose un modèle de risque. Ce chapitre explique comment nous pouvons évaluer le risque et comment nous pouvons l'utiliser dans notre infrastructure de confiance.

Le chapitre 8 détaille un scénario d'application basé sur des transactions du domaine du e-commerce ; nous en fournissons les détails des politiques. Une réalisation du module de négociation et de ses différents composants est présentée.

Le chapitre 9, la conclusion, résume l'apport de la thèse et présente les perspectives de cette étude.

Chapitre 2

Confiance et applications

Dans ce chapitre, nous nous intéressons à la notion de confiance et à la diversité des notions relatives au problème de gestion de la confiance telles que la méfiance, la défiance, la réputation, la recommandation et le risque... Il y a beaucoup de définitions différentes pour la *confiance*, qui dépendent principalement du domaine d'étude et du point de vue de l'auteur sur la conception de son modèle de confiance. Nous nous intéressons à cette notion dans les domaines des sciences humaines et sociales et dans le domaine des applications telles que le commerce électronique (Ebay et autres), la découverte de partenaires potentiels sur Internet, etc.

L'objectif de l'étude présentée dans ce chapitre, sur la diversité de ces notions, est de préciser leur signification dans chaque contexte ou domaine correspondant. Cela nous permet d'éviter l'ambiguïté qui existe toujours entre les définitions existantes. En étudiant la diversité de ces notions, nous donnons également la définition de la confiance du point de vue de la conception de notre infrastructure de confiance.

Nous commençons le chapitre avec la définition de termes qui sont souvent utilisés dans un système de gestion de la confiance, tels que *acteur, rôle, interaction...*. Ensuite, dans la deuxième section, nous discutons sur la diversité de la notion de confiance : *confiance assurée, confiance décidée, familiarité, contexte de confiance, etc.* Dans la suite, les notions *méfiance, réputation, risque...* seront abordées. Nous analyserons également la relation entre ces notions pour préciser les contextes d'application, par exemple la relation entre la confiance et la sécurité, la confiance et le risque...

2.1 Définitions préliminaires

Avant de présenter les notions fondamentales du domaine comme la confiance, le risque et la réputation, nous proposons quelques définitions de termes tels que *acteur*, *credentials* (qualification), interaction, observation, transaction et politique.

Principal

Un principal est une entité autonome et identifiable représentant le sujet à l'initiative de l'action dans le système. Il peut s'agir d'une personne réelle ou d'une personne virtuelle (un programme). Il peut s'engager dans une interaction pour effectuer des actions.

***Acteur* : [*actor*]**

Un acteur (ou une entité) est un membre d'un sous-ensemble du *principal*, qui est constitué seulement par des entités humaines et légitimes (pas par des programmes ou par des machines). Il est représenté par un identifiant (par exemple son nom ou une clé publique). Parfois, nous utilisons également la notion d'*agent* qui est souvent mise en œuvre dans le domaine de l'intelligence artificielle pour décrire cet objet.

Rôle

Terme qui désigne la propriété et la responsabilité d'un acteur dans le système. Exemple : un acteur dans un système de vente en ligne pourrait prendre le rôle de *client* ou celui de *vendeur*.

***Qualifications* [*credentials*]**

C'est un document numérique appartenant à l'entité et qui prouve son identité, une autorisation et/ou ses compétences. Un certificat peut être signé par une autorité de certification qui est censée assurer que le détenteur de ce certificat est bien celui qu'il prétend être ou a bien les qualifications qu'il prétend avoir.

Pour effectuer une transaction, une entité doit prouver au système qu'elle dispose de toutes les qualifications nécessaires pour la mener à bien. Cette entité peut posséder un *credential* qui contient les informations certifiant que son détenteur a le droit de faire telle ou telle action.

***Politique* [*policy*]**

Une politique permet de définir explicitement ce qu'une entité (ou un acteur) est autorisée à faire dans le cadre d'une action donnée et d'un contexte donné. Dans les applications informatiques, un *langage de politique* est souvent utilisé pour exprimer ces autorisations. Le langage de description de politique est une méthode formelle qui permet d'exprimer de manière non ambiguë ces politiques. Ce langage est à destination des utilisateurs du système, et c'est là une des difficultés dans sa conception. Définir *les droits d'accès* aux fichiers sur le système d'exploitation Linux est un exemple simple de politique.

Interaction

Une interaction est une action menée en commun par plusieurs acteurs au sein d'un système. L'exemple le plus simple : deux acteurs dans un système de communication qui échangent des messages. Cette action est considérée comme une interaction.

Transaction

Une transaction spécifie une chaîne d'interactions entre deux acteurs au sein d'un système. En réalité, c'est une séquence d'échanges de messages entre deux acteurs pour aboutir à un but commun (ou plutôt, pour que chacun atteigne son objectif en utilisant des contraintes communes).

Observation

L'observation permet à un acteur de percevoir les événements du système. Un acteur peut disposer d'une chaîne d'observations sur les activités d'un autre acteur (*l'historique*), qui lui permettra d'apporter un éclairage sur les éventuelles futures transactions avec ce même acteur.

Expérience

L'expérience est l'ensemble des comportements d'une entité lors de ses interactions avec d'autres entités du système. Cette expérience est collectée dans l'historique et peut être utilisée pour évaluer les actions futures.

Infrastructure applicative

Un ensemble d'outils (modèle théorique, langage, etc.) servant à la spécification, à la formalisation et à la mise en œuvre des applications dans un contexte spécifique.

Contrôle d'accès

Le contrôle d'accès est une méthode de gestion de l'accès à des ressources. On n'autorise l'accès aux ressources qu'à certaines entités privilégiées.

Authentification

L'*authentification* consiste à vérifier que l'entité qui se présente est bien qui elle prétend être. Cette opération est souvent réalisée en apportant la preuve que l'on connait un secret. L'association d'un mot de passe (secret) et d'un identifiant est la méthode la plus simple.

Autorisation

L*'autorisation* consite à vérifier qu'une entité authentifiée dispose bien des droits nécessaires pour accéder à une ressource.

Certificat

Un *certificat* est un document électronique qui utilise la notion de signature numérique pour associer une clef publique[1] à une identité. L'association clef publique/identité est réalisée sous le contrôle de l'entité qui signe le certificat.

2.2 Confiance

La confiance est une notion que nous appliquons quotidiennement. Nous faisons confiance à tel équipement, à telle personne pour réaliser certaines actions. Dans le dictionnaire Larousse[3], dans un contexte général, *la confiance* est interprétée comme

[1] La notion de clef publique est issue de la cryptographie assymétrique[78]. A une clef publique correspond une clef privée et tout document chiffré par une des deux clefs n'est déchiffrable que par l'autre ; ainsi, une information chiffrée avec une clef privée peut être vérifiée à partir du moment ou l'on dispose de la clef publique correspondante. Si on est sûr de l'identité qui est associée à cette clef publique on est alors sûr que le document à été chiffré par cette personne et cela peut avoir valeur de signature.

« le sentiment de sécurité d'une personne qui se fie à quelqu'un, à quelque chose » en se basant sur ses propriétés (caractère, capacités, expérience, etc.)

Elle a aussi été étudiée dans de très nombreux domaines de recherche tels que les sciences économiques, la philosophie, les relations sociales et les technologies de l'information et de la communication. La définition, dans un contexte général, de cette notion dans ces différents domaines peut se trouver dans le papier de McKnight *et al.* [68], Gerck [34], Lamsal [60] et Corritore *et al.* [18].

Les domaines de recherche sont toujours liés à des perceptions du monde et des méthodes de travail spécifiques. Cela nous donne plusieurs définitions de la confiance, chacune dépendant du domaine.

La confiance représente toujours une relation entre les *acteurs*. Pour une relation de confiance, nous distinguons toujours deux acteurs en présence : le *trustor* (celui qui doit accorder sa confiance) et le *trustee* (celui à qui l'on doit faire confiance).

La notion de confiance peut être interprétée dans des circonstances différentes et de fait, peut se rapporter à des concepts différents. Niklas Luhmann [63] a proposé deux concepts différents : la confiance *assurée* et la confiance *décidée*. Ces concepts sont la traduction de ce que la littérature anglaise désigne par *Confidence* pour la confiance assurée et par *Trust* pour la confiance décidée.

2.2.1 Confiance assurée

Gambetta [32, 33] propose comme définition que « la confiance est *la probabilité subjective* selon laquelle un acteur A s'attend à ce qu'un autre acteur B réalise une action donnée dont dépend son bénéfice ».

On parle de confiance assurée pour décrire le comportement d'un acteur qui anticipe favorablement le déroulement d'une action où il existe d'autres déroulements moins favorables, mais d'une probabilité négligeable. La confiance assurée décrit souvent le cas pour lesquels l'acteur doit seulement envisager un déroulement favorable de l'action, car les cas défavorables ne peuvent pas dépendre de ses choix.

Ainsi, chaque individu au quotidien a toujours une confiance assurée dans certaines actions du fait qu'il doit effectuer cette action. Par exemple, le fait d'aller travailler chaque matin peut avoir un certain nombre de déroulements défavorables (avoir un accident de voiture, recevoir une météorite sur la tête...). Néanmoins, aucun individu ne tiendra compte de ces déroulements défavorables car un refus d'aller travailler n'est pas acceptable. Chacun aura donc une confiance assurée dans le fait de pouvoir arriver vivant au travail.

On parle également de confiance assurée lorsqu'un acteur reconnait une compétence à un autre acteur, et lui autorise de ce fait une action. Par exemple, un malade a une confiance assurée dans le savoir médical de son médecin.

Certains articles parlent de la *fiablité* (reliability) d'un acteur pour exprimer la confiance assurée qui lui est accordée.

2.2.2 Confiance décidée

McKningt et Chervany [68] ont suggéré que « la confiance est *la volonté* d'un acteur de dépendre d'un autre acteur (de quelqu'un ou de quelque chose) dans un contexte donné où il a le sentiment d'être en sécurité, même dans le cas où une conséquence négative pourrait se réaliser ».

Cette définition se réfère à la notion de *dépendance*. La décision d'un acteur pourrait dépendre d'un autre acteur, avec ou sans indication de confiance. Cela signifie que la

décision est prise dans un contexte incertain. Un bon exemple relatif à cette définition est le fait de demander notre chemin à un étranger. Nous ne savons pas exactement si cette personne nous indique la bonne direction ou si elle nous donne une indication erronée, mais nous suivons quand même ses indications parce que dans ce cas précis c'est le seul choix que nous pouvons faire. Autrement dit, nous prenons la décision sachant qu'un risque existe.

2.2.3 Contexte de la confiance

Dans une relation de confiance, le contexte est important. Ce *contexte* indique les événements de la situation où la relation est établie. La relation de confiance doit toujours être définie dans un *contexte donné* car hors de ce contexte, la relation peut ne plus être affirmée.

Nous pouvons donner des exemples concrets qui illustrent ce fait. *Alice* peut faire confiance à *Bob* pour lui donner des cours d'informatique, mais elle pourrait ne pas lui faire confiance pour garder ses enfants. Un autre exemple : *Bob* utilise le service *PayPal* pour ses transactions en ligne ; il peut faire confiance à *PayPal* pour des transactions de moins de 100€, mais peut-être ne lui fera-t-il pas confiance pour des transactions dont la somme engagée est plus élevée.

2.2.4 Familiarité

Gambetta [32, 33] et Niklas Luhmann [63] ont également abordé la notion de *familiarité* (familiarity). On peut définir une familiarité comme étant un fait que nous avons observé à de nombreuses reprises et que nous savons inéluctable au moment présent. La familiarité représente donc un niveau maximal de confiance ; nous ne le remettons pas en cause.

Les différents niveaux de confiance peuvent se représenter sur une ligne graduée de 0 à 1. Le niveau de confiance 0 est comparable à une paranoïa (dans la mesure où l'acteur refusera résolument d'accorder sa confiance). Entre le 0 et 1 viennent la confiance décidée, puis la confiance assurée. Au niveau de confiance 1, nous trouvons la familiarité.

2.2.5 Confiance et domaines d'application Internet

Nous nous intéressons à la notion de confiance appliquée à l'Internet ce qui nous aidera à présenter notre système de gestion de la confiance par la suite. Plusieurs définitions de la confiance sont proposées dans ce domaine

Dans sa thèse, Tyrone Grandison [36] considère que « la confiance est *la croyance* mesurée par un *trustor* en ce qui concerne la compétence, l'honnêteté, la sécurité et la fiabilité d'un *trustee* dans *un contexte donné* ». La définition souligne le fait que la confiance est une *croyance* et qu'elle est subjective. À ce titre, elle doit toujours être envisagée dans un contexte spécifique.

Dans ce contexte d'étude des applications Internet, nous considérons cette définition comme une base pour construire notre modèle de confiance. De notre point de vue, la confiance est *la croyance* du *trustor* en le *trustee*, mesurée en utilisant toutes les sources d'informations disponibles concernant les acteurs et leurs actions : l'expérience, les informations de réputation, celles du risque... Comment mesurons-nous la confiance entre les acteurs dans un système ? Ceci est l'un des buts du système de gestion de la confiance que nous allons présenter par la suite.

2.2.6 Ambiguïté sur la notion de confiance

Depuis l'apparition de la notion de gestion de la confiance et des systèmes de gestion de la confiance, le terme confiance est compris de façons différentes. Parfois cela rend ambiguë cette notion et nous devons la préciser.

Ce terme *confiance* est utilisé implicitement comme la notion de *contrôle d'accès, d'authentification, d'autorisation ou de délégation* dans le mécanisme de confiance proposé par Blaze *et al.* [64, 23, 30]. Le terme *confiance* est utilisé dans le contexte suivant : supposons que le système contienne deux acteurs A et B et que l'acteur B veuille effectuer une action en utilisant des ressources gérées par l'acteur A. La question qui se pose pour A est de savoir *si il peut faire confiance à* B ? Ceci lui permettrait d'effectuer l'action qui lui est demandée. Autrement dit, il nous faut déterminer le niveau de la relation de confiance entre A et B.

Pour être autorisé, B doit présenter à A des qualifications (*credential* en anglais, un certificat par exemple) qui contiennent des informations sur sa validité, l'action à effectuer, l'autorité qui l'a signée... L'acteur A en vérifie la validité et après avoir vérifié auprès de son autorité s'il disposait des autorisations nécessaires, décide de donner ou pas à B un droit de contrôle ou d'accès aux ressources demandées. Évidemment dans ce contexte, la relation de confiance est définie comme une suite d'actions *de contrôle d'accès, d'authentifications et d'autorisations.*

Lorsque l'on étudie la littérature qui concerne les mécanismes de confiance, nous y découvrons que le mot *confiance* est souvent employé de manière inappropriée à la place *de contrôle d'accès*, *d'authentification* ou d'*autorisation.* Il nous a semblé important de correctement distinguer ces différentes notions pour lever toutes ambiguïtés.

Voici quelques exemples pour faire apparaître une distinction claire entre la *confiance* et les autres notions (*contrôle d'accès, autorisation et authentification*) :

- *Confiance/contrôle d'accès* :
  A fait confiance à B pour la garde de son ordinateur portable, mais A ne donne pas à B le mot de passe qui lui permettrait d'utiliser cet ordinateur. C'est à dire que la confiance n'implique pas le contrôle d'accès et vice versa. Il y a aussi des situations où le contrôle d'accès est accordé mais la confiance n'est pas établie. A donne à B le mot de passe pour utiliser son ordinateur portable en tant qu'invité, pour accéder à l'Internet, mais A n'est pas sûr que B n'essayera de faire d'autres choses avec son ordinateur.
- *Confiance/Autorisation* :
  L'autorisation peut être considérée comme la conséquence de la relation de confiance. A fait confiance à B et il peut lui donner l'autorisation d'utiliser son ordinateur avec tous les droits, en lui donnant un contrôle d'accès à tous les services, y compris le doit d'administrateur.
- *Confiance/Authentification* :
  L'authentification peut être expliquée comme la vérification d'identité d'une entité. Cette vérification est réalisée soit par les données *utilisateur/mot de passe*, soit par l'utilisation d'un service d'authentification ou par l'utilisation de certificats. Cela signifie que nous ne pouvons pas confondre les termes *confiance* et *authentification.*

2.2.7 Méfiance et défiance

Nous avons présenté de manière assez détaillée la confiance dans les sections précédentes. Il y a une autre notion qui a beaucoup d'importance lorsque l'on parle de

confiance : il s'agit de la *méfiance*. Le terme *méfiance* exprime l'inverse de la notion de *confiance*. Dans [36, 35], Grandion et al. proposent la définition suivante : « la méfiance est *la croyance* mesurée par un *trustor* en ce qui concerne l'incompétence, la malhonnêteté, l'insécurité et le manque de fiabilité d'un *trustee* dans *un contexte donné* ».

Nous considérons que la *méfiance* de l'acteur A envers l'acteur B est la *conséquence négative* de l'évaluation de la confiance de A en B en utilisant toutes les sources d'informations possibles : les certificats, l'expérience, l'information liée au risque...

La *défiance* représente un autre aspect de la confiance que s'accordent deux entités. Cette notion représente la situation où l'on a un a priori défavorable sur le déroulement de l'action (il y a des très grandes chances que cela se finisse mal). Notre interlocuteur a acquis une sorte de confiance négative.

2.3 Risque

Le risque apparait souvent de manière implicite dans nos actions de tous les jours. Quand nous entreprenons une action dans une situation incertaine, notre décision est toujours prise en sachant qu'il existe un risque.

En général, le risque est compris comme une perte potentielle, identifiée et quantifiable, inhérente à une situation ou une activité, associée à la probabilité de l'occurrence d'un événement ou d'une série d'événements.

Le risque représente ce qui peut arriver lorsqu'un acteur (ou un système) décide d'accorder à tort sa confiance à une entité. Cela peut être vu comme l'évaluation des conséquences de l'action entreprise lorsque la transaction s'est déroulée au détriment de son bénéficiaire.

2.3.1 Définitions

Le notion de risque est présente dans de nombreux domaines et on trouve des définitions du risque souvent différentes dans chacun de ces domaines.

Dans la norme ISO [46] sur la gestion de risque, il est dit que *« le risque est la combinaison de la probabilité d'occurence d'un événement et de ses conséquences »* . Pour les sciences, on considère que *« le risque est l'espérance mathématique d'une fonction de probabilité d'événements »* . En économie et en finance, le risque porte sur les actifs financiers et on considère que le risque est *« une possibilité de perte monétaire due à une incertitude que l'on peut quantifier »* .

Dans le domaine de l'informatique, on parle plus facilement du risque en termes de menaces et de vulnérabilités liées au système informatique. Et bien sur, l'informatique peut être utilisée pour tenter d'identifier et d'évaluer des risques. Le projet SECURE [15] en est un bel exemple.

2.3.2 Évaluation du risque

Pour évaluer le risque, nous devons considérer l'incertitude qui est attachée aux conséquences de chaque action entreprise. Il s'agit d'un travail difficile à réaliser mais pourtant indispensable si l'on veut construire une application ou un système qui prend en compte ce facteur.

Le risque sera évalué avec l'aide d'un *modèle de risque* adapté à chaque système. Avec ce modèle, nous pourrons déterminer les conséquences et leurs probabilités d'occurence

pour pouvoir l'évaluer par la suite. Les *modèles de risques* seront abordés au chapitre 3.

2.3.3 Risque et confiance

Nous voyons donc que les notions de risque et de confiance sont étroitement liées. Pour une situation ou une action à entreprendre, s'il n' y a pas de risque il n'est donc pas nécessaire de faire confiance. Autrement dit, la question de confiance ne s'est pas posée.

Par contre, dans la situation où le risque est identifié, on se sert souvent de l'évaluation des risques pour définir le niveau de confiance que l'on peut accorder lorsque l'on doit entreprendre une action.

La confiance est difficile à évaluer objectivement par un processus de décision. Pourtant, risque et confiance entretiennent des liens étroits, nous pouvons utiliser la notion de risque comme le dual de la confiance. Le risque peut être évalué au travers du coût des conséquences de l'action entreprise.

2.4 Réputation

En général, la réputation est l'opinion d'une communauté envers une personne, un groupe, ou une organisation. C'est un facteur important dans de nombreux domaines, tels que l'éducation, le commerce ou le statut social. Cette notion fait toujours référence à une communauté.

Dans [54], Josang *et al.* envisagent que « la réputation est ce que l'on dit ou croit d'une personne ou des propriétés d'un objet ». Ils pensent également que la réputation est un moyen pour renforcer la confiance. Dans une communauté, on peut faire confiance à quelqu'un s'il a bonne réputation.

Selon Abdul-Rahman et Hailes [2], la réputation est une estimation du comportement d'une entité dans la communauté, basée sur ses comportements passés. Cette définition prend en compte deux sources d'informations principales pour estimer la réputation : les expériences passées avec le sujet et les *recommandations* des autres entités de la communauté.

Dans les systèmes de réputation mis en œuvre par les sites de vente et d'enchères en ligne tels que Amazone [37] ou eBay[38], la réputation d'un acteur est interprétée implicitement comme *l'évaluation de ses comportements passés*. La réputation des produits et des acteurs peut être calculée de la manière suivante :

- pour un produit : chaque fois qu'un produit est vendu, l'acheteur peut évaluer sa qualité sur une échelle de 1 à 5 étoiles. La réputation du produit dans le système sera déterminée par une valeur moyenne de toutes ces évalutations.
- pour un utilisateur : après avoir achevé une transaction il peut évaluer la manière dont la transaction s'est déroulée en spécifiant un degré de satisfaction ; par exemple *positif, neutre, négatif*. Ces évaluations sont associées au profil du correspondant de cet utilisateur. Le nombre de ces évaluations nous donnera une image de la réputation de cet utilisateur dans le système.

2.5 Recommandation

Abdul-Rahman et Hailes [2] précisent que *"la recommandation est un avis échangé sur la fiabilité d'une entité tierce"*. Cette valeur de recommandation pourrait être qua-

litative (*"positive"* ou *"négative"*) ou quantitative (une valeur mesurée par une échelle numérique, par exemple les valeurs $+1$, $+2$,..., $+5$). Le choix de la représentation de la valeur de la recommandation est dépendant de chaque système.

Considérons un système distribué constitué de plusieurs entités qui collaborent entre elles. Chacun peut évaluer la qualité et la fiabilité d'une collaboration avec une entité tierce et fournir cette information à une troisième entité. A l'échelle du système, l'ensemble de ces actions permet de construire un mécanisme d'échange et de propagation des informations de réputation entre les différentes entités du système. Cela a permis de construire un système de *recommandation*.

Une recommendation peut être *directe* ou *indirecte*. Une recommandation directe est une recommandation que fait une entité A à une entité C au sujet d'une entité B (A recommande B à C). Une recommandation indirecte est à interpréter comme étant l'avis que donne publiquemnt A au sujet de B. Dans ce second cas, toutes les entités du système peuvent prendre connaissance de l'avis de A sur B. Les sites Amazon [37] ou eBay[38] sont de bons exemples de recommandations indirectes : les utilisateurs donnent leur avis publiquement sur les autres utilisateurs (ou produits) concernant leurs transactions, tout le monde peut utiliser ces informations si nécessaire.

Recommandation et réputation sont étroitement liées et comme nous l'avons vu dans la section précédente, la recommandation est la base des calculs de nombreux mécanismes de réputation.

2.6 Confiance et sécurité

La confiance et la sécurité sont étroitement liées et les processus de décision de la confiance traditionnels (que nous présenterons dans le chapitre suivant) comme KeyNote [64], TrustBuilder [26] traitent de manière identique ces deux notions.

Il existe une relation stricte entre les notions de *sécurité* et de *confiance*. Si la transaction que nous allons entreprendre peut se faire en toute sécurité, alors nous considérons que nous ne prenons aucun risque et le problème de la confiance ne se posera donc pas. Il est nécessaire de se préoccuper des problèmes de confiance dès lors que la sécurité des relations n'est plus assurée.

Dans le cas général, le mécanisme de sécurité protège les ressources contre les accès malveillants par un contrôle qui en restreint l'accès aux personnes autorisées. Mais nous devons aussi nous protéger contre la partie qui nous offre le service ou les ressources car ce fournisseur de services pourrait lui aussi nous donner des informations erronées dans le but de nous tromper.

2.7 Gestion de la confiance

Le terme **gestion de confiance** *[trust management]* a été introduit initialement par M. Blaze *et al.* en 1996 [64]. Les auteurs ont considéré que la gestion de confiance est « une approche unifiée pour spécifier et interpréter des politiques, des qualifications et des relations permettant d'autoriser ou non des actions ».

En analysant les systèmes de gestion de confiance basés sur cette définition tels que PolicyMaker [64, 12] ou KeyNote [65], nous pouvons voir que la gestion des accès et des actions est construite sur un mécanisme à base d'autorisations. Supposons qu'il y ait deux acteurs A et B dans le système. L'acteur B veut entreprendre une action auprès de l'acteur A. Pour ce faire B doit lui présenter les qualifications nécessaires. L'acteur A

fera confiance à B si celui-ci lui présente des certificats et des qualifications valides pour l'action (certificats signés par une autorité de certification reconnue et autorisations d'agir signés par les acteurs autorisés à le faire).

La gestion de la confiance telle qu'elle est abordée par les auteurs de KeyNote ne fournit pas un point de vue complet sur ce que doit être à notre sens un système de gestion de la confiance. Nous sommes ici face à un mécanisme de délégation d'une confiance qui est déjà acquise : les entités ont une confiance *assurée* dans celle à qui elles ont décidé de déléguer la gestion de certaines prérogatives et ces entités ne peuvent offrir plus que ce qui leur est autorisé.

Une définition plus large et plus complète de la notion de *gestion de la confiance* a été proposée par T. Grandison *et al.* [36]. Les auteurs supposent que *« la gestion de la confiance est l'activité de rassembler, de codifier, d'analyser et de présenter les preuves concernant la compétence, l'honnêteté, la sécurité et la fiabilité d'un acteur en vue de prendre des décisions »* .

Les *preuves* qui sont utilisées ici sont des qualifications (des certificats dans ce cas précis), des évaluation du risque, l'expérience et la recommandation. Toutes ces informations concernant une action sont utilisées pour prendre les décisions, pour valuer certains critères liés à la relation de confiance ou ré-évaluer une relation de confiance existante.

Avec de telles spécifications, cette vison de la gestion de la confiance est plus proche de l'idée que nous nous en faisons dans le monde réel. Il nous est possible de voir comment, étape après étape, la relation de confiance est établie.

Ces deux définitions de la confiance correspondent à des approches bien différentes : *l'une est basée sur la définition de politiques et l'autre sur la réputation des participants.* La première approche utilise uniquement un mécanisme de délégation basé sur la présence de qualifications pour définir les politiques de la confiance (définir qui peut faire quoi) alors que la seconde approche est plus souple et plus proche du monde réel en utilisant des sources d'informations différentes dans les raisonnements qui permettent d'évaluer les relations de confiance.

Dans le chapitre suivant, nous allons présenter les principales approches développées dans la littérature pour les infrastructures de gestion de la confiance.

2.8 Négociation de la confiance

Une autre notion qui est souvent utilisée dans ce domaine est la notion de *négociation de confiance* (trust negociation). Elle a été abordée initialement par K.E. Seamons *et al.* avec TrustBuilder [83]. Du point de vue de ce *framework*, la négociation de la confiance est vue comme un processus d'échange de données entre deux parties pour établir la confiance afin de réaliser une transaction par la suite. Ce concept a ensuite été abordé dans les travaux de E. Bertino *et al.* avec *Trust-X* [5, 6], avec le framework de confiance *Role-based Trust Management* de Ninghui Li *et al.* [62].

Tous ces frameworks sont basés sur des mécanismes de délégation pour l'autorisation des actions. Les processus de négociation utilisés par ces frameworks peuvent être considérés comme des protocoles de sécurité. Le concept de *négociation de confiance* n'a donc pas été utilisé dans son sens usuel : si le protocole fonctionne normalement, la sécurité de la transaction est assurée, de ce fait le problème de confiance ne se pose pas. Avec ces premiers travaux sur la gestion de la confiance, nous voyons que les notions de confiance et de sécurité sont imbriquées : la confiance est la conséquence de la

vérification de la sécurité, ce qui veut dire que quand la sécurité est avérée, la confiance est acquise. Nous sommes dans une situation où il n'y a pas de confiance sans sécurité, ce qui est en contradiction avec ce que nous affirmions précédemment : la confiance est nécessaire lorsque la sécurité ne peut plus être assurée.

Depuis ces travaux précurseurs, plusieurs approches de la gestion de confiance ont abordé le sujet soit par l'utilisation des politiques de sécurité, soit par l'utilisation de la réputation.

2.9 Synthèse

Dans ce chapitre, nous avons présenté les notions de base utilisées par les systèmes de gestion de la confiance : la confiance, la réputation, la recommandation, le risque... Pour chaque terme, nous avons d'abord présenté les différentes définitions existant dans la littérature, proposé une brève analyse et enfin donné une définition qui sera utilisée dans la conception de notre infrastructure de gestion de la confiance. Nous avons précisé tous ces termes afin d'éviter toute ambiguïté.

Chapitre 3

Gestion de la confiance

Dans ce chapitre, nous présentons différentes approches de la littérature représentatives des mécanismes de gestion de la confiance : les approches basées sur l'utilisation de politiques, les approches basées sur l'expérience des participants et les approches hybrides. Pour chacune, nous présentons les principaux systèmes existants que sont KeyNote [64], SULTAN [36], le role-based trust management [62], TrustBuilder [84, 26] et le framework SECURE [15]. À la fin de ce chapitre, nous effectuons une analyse générale de ces systèmes et abordons les principales idées qui sont à la base de notre solution de gestion de la confiance.

3.1 Les approches de gestion de la confiance

Une infrastructure de gestion de la confiance est nécessaire à de nombreuses applications comme nous l'avons montré au chapitre 1. Mais de quoi est exactement constitué un système de gestion de la confiance ? Quelles sont les fonctionnalités qu'il doit fournir et dans quels contextes nous est-il possible de proposer une solution permettant d'établir une situation de confiance ?

L'expression « système de gestion de la confiance » (*Trust Management System*) désigne une solution logicielle de prise en compte des problèmes liés à la confiance au sein des applications. Ces solutions sont apparues assez récemment et PolicyMaker [64] fut l'un des précurseurs. Notons que la plupart de ces systèmes ne sont pas des logiciels indépendants mais des bibliothèques ou des services dont l'objet est de fournir aux applications le moyen d'évaluer la confiance avant d'entreprendre certaines actions.

En étudiant l'état de l'art concernant les premiers systèmes de gestion de la confiance, nous pouvons distinguer deux familles d'approches principales : les approches basées sur l'utilisation de politiques et les approches basées sur l'utilisation de la réputation. Une troisième approche, dite *approche hybride* apparaît dans les solutions récentes de gestion de la confiance. C'est une approche hybride qui est mise en œuvre dans le cadre de notre solution.

3.2 Gestion de la confiance basée sur la politique

Cette approche vise les applications qui ont besoin d'un mécanisme de contrôle d'accès et d'autorisation. Le principe de ce mécanisme est d'utiliser un langage dédié à l'expression des politiques pour spécifier les règles décrivant le contrôle d'accès aux ressources, le raisonnement sur ces règles permettant d'établir la confiance entre les parties

impliquées dans le système. Le principal but de ce raisonnement est de déterminer si un utilisateur inconnu peut être considéré comme de confiance en fonction des politiques à satisfaire et des *credentials* présentés.

En général, un système de gestion de la confiance basé sur la politique se compose des éléments suivants :

- Les actions : le rôle du système est de décider d'autoriser ou non ces actions.
- Les acteurs : ils vont demander ou accorder des permissions nécessaires à l'exécution d'une action.
- Les politiques : elles permettent de spécifier les autorisations des acteurs ou des actions. Ces politiques sont souvent décrites par des règles (*assertions*).
- Les délégations : ce sont des clauses qui permettent à certains acteurs d'autoriser d'autres acteurs à effectuer des actions qui leur étaient autorisées.
- Un moteur de raisonnement sur la confiance : il s'agit d'un service qui analyse et traite pour le compte des différents acteurs, les assertions conformément à sa politique.

Le principe général de fonctionnement du système : une application qui, pour le compte d'un acteur donné, veut exécuter une action (accéder à une ressource protégée, exécuter une commande...), envoie une demande d'autorisation au système. Le moteur de raisonnement de ce système réalisera alors une vérification de cette demande en se basant sur l'ensemble des assertions de la politique, de l'action requise, des informations de délégation et des crédentials présentés. Le résultat de cette opération permettra à l'application de prendre la décision d'accéder ou non à la ressource demandée.

En général, les systèmes de gestion de confiance de ce type sont utilisés par les systèmes qui ont besoin d'un niveau de protection fort comme les applications de paiement ou d'authentification. Les principaux systèmes existants de ce type sont KeyNote [64], TrustBuilder [84, 26], Role-based trust management [62] et SD3 [48]. Il sont présentés en détail dans les sections suivantes.

3.2.1 Keynote

KeyNote est sans doute le premier système de gestion de la confiance présenté comme tel. Entre autre, il introduit la notion de "Trust Management System". Il a été créé par M. Blaze *et al.* [8, 10, 9, 11]. C'est un système de gestion de la confiance représentatif des approches basées sur l'utilisation de politiques. KeyNote a été construit de manière à faciliter son intégration et son utilisation dans une vaste gamme d'applications. KeyNote a été normalisé par la *Request For Comments (RFC)* 2704 [9]. Il est utilisé aujourd'hui dans l'implémentation IPSEC de OpenBSD.

KeyNote fournit un langage de description de politiques qui permet à chaque participant de définir ses propres politiques, actions et *credentials* sous forme d'*assertion*. Il fournit également un moteur de raisonnement simple et robuste pour vérifier que les actions entreprises par les acteurs et les *credentials* fournis sont conformes à la politique du service demandé.

Voici un exemple d'*assertion* :

```
KeyNote-Version: 2
Comment:         exemple d'assertion
Local-Constants: USER = "vu"
                 ROOT = "root"
Authorizer:      ROOT
```

```
Licensees:      USER
Conditions:     password != "test" -> _MAX_TRUST ;
                password == "test" -> _MIN_TRUST ;
Signature:      BDANBgkqhkiG9w0...
```

Les différents champs de l'assertion ont la signification suivante :
`KeyNote-Version` : version de KeyNote utilisée ;
`Comment` : commentaires associés à l'assertion ;
`Local-Constants` : définitions de variables locales à l'assertion ;
`Authorizer` : identification de l'acteur qui délègue une prérogative (propriétaire de la clef publique) ;
`Licensees` : identification des acteurs qui sont autorisés à entreprendre l'action. Si ce champ n'apparaît pas, KeyNote considère que toute clef est valable pour cette assertion ;
`Conditions` : description des conditions que l'acteur doit remplir pour avoir le droit d'entreprendre l'action ;
`Signature` : signature numérique de l'assertion par l'`Authorizer`. Cette signature permet à KeyNote de vérifier la validité de l'assertion.

Le fonctionnement de KeyNote est exprimé dans la figure 3.1 :

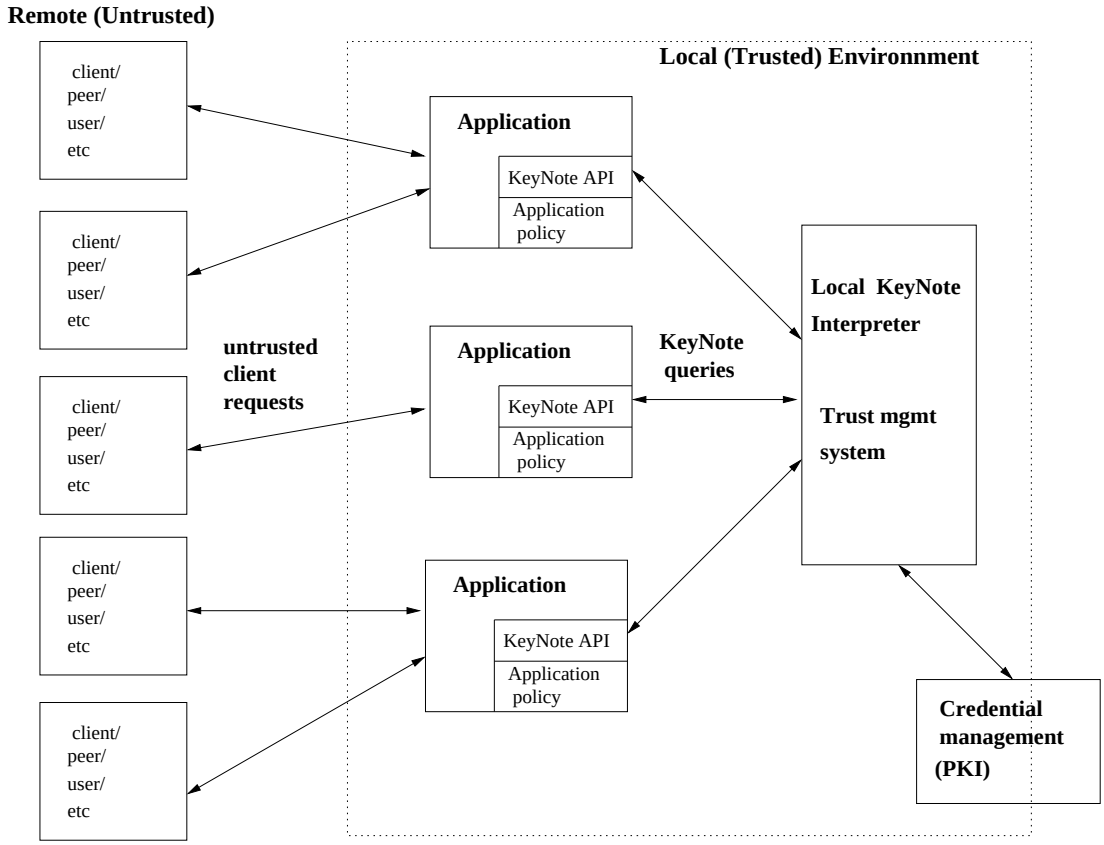


FIG. 3.1 – KeyNote

KeyNote fournit un utilitaire en mode ligne de commande (pour la réalisation de tests limités) et une API d'accès complète. L'API de KeyNote offre toutes les fonc-

tions nécessaires à l'enregistrement des assertions et à la soumission de requêtes. Un mécanisme de sessions permet même une utilisation concurrente.

En conclusion, nous pouvons voir que le principal intérêt de KeyNote est bien la puissance d'expression du langage des politiques. Ce langage est bien défini et il permet d'exprimer précisément les *credentials*, les délégations, les politiques et les actions. De plus, il dispose d'une API qui permet d'intégrer facilement ce mécanisme de confiance aux applications existantes. Pourtant, le langage ne supporte pas de mécanisme de protection de la confidentialité des données sensibles, par exemple une assertion peut contenir des informations sensibles (mot de passe dans l'exemple précédent). Il ne propose pas non plus de mécanisme de négociation pour l'établissement de la confiance entre les acteurs.

3.2.2 Framework TrustBuilder

Trustbuilder est un framework de négociation automatique de la confiance. Ce framework gère le problème de l'établissement de la confiance entre les entités au moyen d'un échange de credentials. Un *credential* est une assertion signée par son émetteur et qui contient les prérogatives accordées à son propriétaire.

La négociation est un processus d'échange de credentials entre les deux participants. Chaque participant a ses propres *credentials* et ses propres politiques d'accès au service. Chaque participant est associé à un agent de sécurité qui gère pour son compte la découverte, l'échange, la vérification des credentials et la politique de sécurité. La figure 3.2 présente le fonctionnement du framework.

Pour spécifier une politique, le framework utilise **PAL** (Property-based Authentication Language) et **RAL** (Roled-based Authorization Language) qui sont deux langages proposés par IBM Trust Establishement [44, 43]. L'algorithme de découverte de la chaîne de credentials permet de finaliser la négociation.

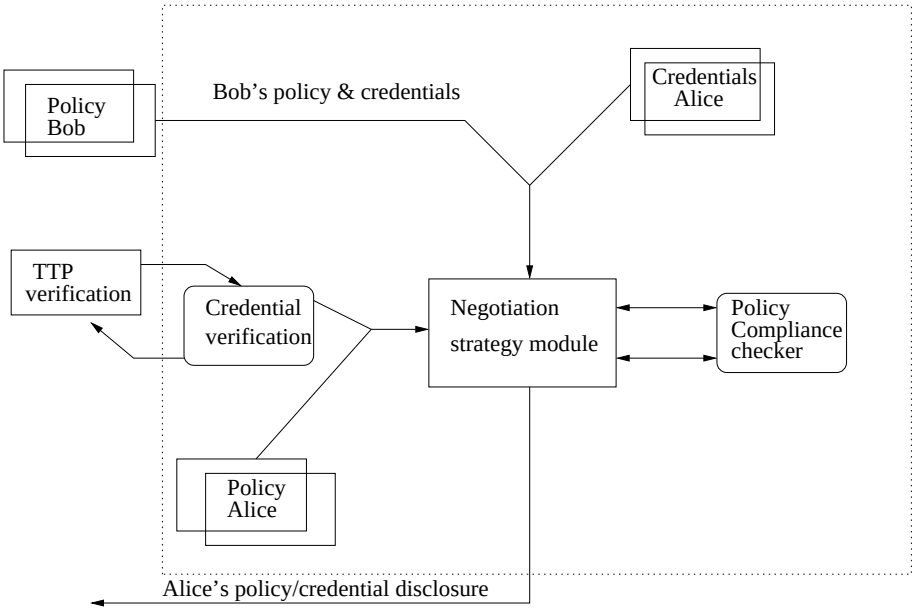


FIG. 3.2 – Trustbuilder framework

3.2.3 RT : Role-based Trust-management Framework

RT [62, 61], développé par Ninghui Li et J. C. Mitchell est aussi un framework de gestion de la confiance basé sur une approche utilisant des qualifications. Dans ce système, chaque entité désigne un individu ou un processus. Elle peut délivrer des qualifications ou faire des requêtes. Un *rôle* définit un ensemble d'entités à qui sont associés des attributs particuliers.

Les ressources sont gérées avec un modèle de contrôle d'accès basé sur les attributs (ABAC). Le système fournit les mécanismes nécessaires pour associer des attributs aux entités ; déléguer des attributs à d'autres entités ; raisonner sur ces attributs... Les qualifications sont constituées à partir de ces attributs.

Une entité dispose d'un attribut donné si elle appartient au rôle correspondant. Le système RT est composé des éléments suivantes :

- RT_0 : le composant de base du système, il permet de définir les *rôles*. Notons qu'un rôle défini par RT_0 n'a aucun paramètre. Pour un principal K_A et R un rôle, la forme $K_A.R$ définit un nom de rôle dans RT_0. Nous avons également les définitions suivantes :
  - *membre simple* : $K_A.R \leftarrow K_D$, définit le principal K_D appartenant au rôle $K_A.R$.
  - *contenant de rôle* : $K_A.R \leftarrow K_B.R_1$, définit le rôle $K_A.R$ contient le rôle $K_B.R_1$. Cela veut dire que tous les membres de $K_B.R_1$ appartiennent également au rôle $K_A.R$.
  - *contenant d'intersection* : $K_A.R \leftarrow K_{B_1}.R_1 \cap \ldots \cap K_{B_k}.R_k$, signifie que le rôle $K_A.R$ contient l'intersection de tous les rôles $K_{B_1}.R_1 \ \ldots \ K_{B_k}.R_k$
  - *délégation simple* : $K_A.R \Longleftarrow K_B : K_C.R_2$ ($K_C.R_2$ est optionnel), signifie que K_A délègue ses autorités sur R au principal K_B. Si $K_C.R_2$ est présent, cela veut dire que $K_A.R$ délègue ses autorités au K_B et le K_B ne peut qu'attribuer ces autorités aux membres de $K_C.R_2$.
- RT_1 : définir les rôles avec des paramètres.
- RT_2 : ajouter à RT_1 les objets logiques pour regrouper les permissions. Cela permet de raisonner sur les attributs.
- RT^T : permet de grouper quelques entités pour traiter une tâche commune sur les rôles
- RT^D : fournit un mécanisme de délégation pour l'activation des rôles entre les principals.

Voici un exemple adapté de [61] pour illustrer le fonctionnement de ce système : le service d'accueil de IEEE, *I3EA*, ne donne une subvention (*grant*) qu'aux participants qui sont à la fois membres de IEEE et qui ont un article à la conférence POLICY 2010. Les qualifications suivantes prouvent que *Alice* est éligible pour cette subvention.

$$C_1 : I3EA.grant \leftarrow Policy2010.participant \cap IEEE.member$$
$$C_2 : Policy2010.participant \leftarrow Alice$$
$$C_3 : IEEE.member \leftarrow Alice$$

Dans cet exemple, C_2 et C_3 indiquent que le rôle *Alice* appartient aux rôles $Policy2010.participant$ et $IEEE.member$, C_1 indique que l'intersection des $Policy2010.participant$ et $IEEE.member$ appartient au rôle $I3EA.grant$.

Les composants de ce système sont les éléments d'une logique modale. Nous pouvons définir la politique de chaque entité en lui affectant des rôles et en définissant ses

qualifications. Les raisonnements sur les qualifications et les délégations de ces qualifications permettent de construire des relations entre les entités. Ces relations ont pour but d'établir la confiance entre les entités.

3.2.4 SD3

SD3 (Secure Dynamically Distributed Datalog) proposé par T. Jim [48] est un système de gestion de la confiance qui utilise la logique pour représenter la politique. Le système se compose d'un langage de politique, d'un évaluateur local de cette politique et d'un mécanisme de collecte (retrieval) de certificats. Le langage de politique est une extension de *datalog* [66], un langage de requête dédié aux bases de données déductives.

Un *scénario* définit en SD3 applique un ensemble des règles de la forme suivante :

```
T(x,y) :- K$E(X,y)            ..... (1)
T(x,y) :- (K@A)$E(X,y)        ..... (2)
```

`T(x,y)` indique la confiance que x a en y. La règle (1) indique que l'assertion `T(x,y)` est vérifiée lorsque `K$E(X,y)` est vérifiée ; `K$E(X,y)` est la relation `E(x,y)` sous le contrôle de la clé K (le prédicat est signé par la clé K). L'opérateur @ sert à identifier un nom global avec une adresse IP avec le syntaxe (K)$E : cela identifie la relation E de K, à l'adresse IP A. Par conséquence, la règle (2) indique que `T(x,y)` est vérifiée si `E(x,y)` est vérifiée par la machine à l'adresse IP A avec la clé K. De par ses propriétés, ce langage permet de définir des *chaînes de confiance* distribuées.

Le *scénario* est intégré dans le programme. L'évaluation est réalisée en donnant à ce programme une requête et un certificat. L'évaluateur donnera une réponse pour cette requête ainsi qu'une preuve indiquant qu'elle respecte la politique de sécurité. Nous pouvons remarquer que le mode de fonctionnement de SD3 est assez proche de celui du système KeyNote.

3.2.5 Framework de confiance IBM

Le framework de confiance IBM [44, 43] considère que l'établissement de confiance est un composant du *e-commerce*. La confiance nécessaire à la transaction de *e-commerce* sera vérifiée en utilisant des certificats. Un certificat est remis à chaque entité du système pour chaque rôle particulier qu'elle a à jouer. IBM a développé un modèle de contrôle d'accès basé sur les rôles qui utilise les certificats. Il se compose d'un module d'*établissement de confiance* (en Java) et d'un langage de politique : TPL (Trust Policy Language). Les certificats *X.509 v3* [42] sont utilisés par défaut.

Le module d'établissement de confiance valide le certificat du client et lui affecte le rôle spécifié par le certificat. La politique locale exprimée en TPL définit ce qui est admis pour ce rôle. La rédaction des politiques en TPL utilise une syntaxe XML qui est présentée dans [44, 43]. La structure primitive de TPL est le *groupe* et pour chaque groupe, il y doit y avoir une règle gérant les membres de ce groupe.

```
<POLICY>
   <GROUP NAME="self">
   </GROUP>
   <GROUP NAME="partners">
      <RULE>
         <INCLUSION ID="partner" TYPE="partner" FROM "self">
         </INCLUSION>
```

```
      </RULE>
   </GROUP>
   <GROUP NAME="departments">
      <RULE>
         <INCLUSION ID="partner" TYPE="partner" FROM="partners"
         </INCLUSION>
      </RULE>
   </GROUP>
   <GROUP NAME="customers">
      <RULE>
         <INCLUSION ID="customer" TYPE="employee" FROM="departments">
         </INCLUSION>
         <FUNCTION>
            <GT>
               <FIELD ID="customer" NAME="rank"></FIELD>
               <CONST>3</CONST>
            </GT>
         </FUNCTION>
     </RULE>
   </GROUP>
</POLICY>
```

Dans cet exemple, le premier groupe désigne la racine de la certification. Le deuxième groupe indique que les entités ayant des certificats de type `partner`, signés par la racine, font partie du groupe `partners`. Le groupe `departements` est composé des entités qui ont un certificat de type `partner`, signé par le groupe `partners`. Le dernier groupe, `customers`, est composé des utilisateurs ayant un certificat de type `employee` signé par un membre du groupe `departements` et un classement plus grand que 3 (valeur de l'attribut `rank`). Le module d'établissement de confiance détermine si un rôle particulier est affecté à l'entité, et ensuite vérifie que l'accès à la ressource ou au service demandé est autorisé. Nous pouvons constater que toutes les information concernant l'entité sont accessibles et que cela peut poser des problèmes de protection des données personnelles (*privacy*).

3.3 Gestion de la confiance basée sur l'expérience

Le système comporte plusieurs entités (ou acteurs) et est de type *peer-to-peer*. Ces systèmes ont été largement étudiés ces dernières années. Le principe de cette approche est que l'on utilise des données sur le comportement passé de l'acteur et les recommandations des autres entités pour évaluer la confiance que l'on peut attribuer à cet acteur.

La relation entre deux acteurs, le *donneur de confiance* (trustor) et le *demandeur de confiance* (trustee), s'établit de la manière suivante : le donneur de confiance aura pour rôle d'estimer le niveau de confiance (*trust level*) qu'il peut accorder au demandeur de confiance en se basant sur sa réputation ; il décidera alors de lui faire confiance ou non en fonction du résultat obtenu. Cette réputation est basée sur les relations passées entre donneur et demandeur de confiance (expérience) et parfois sur le partage d'expériences avec d'autres demandeurs de confiance (recommandations).

Ces systèmes se sont beaucoup développés depuis l'apparition des sites d'enchères en ligne : le donneur de confiance est le site (ou éventuellement un ensemble de sites partenaires), les demandeurs de confiance sont les utilisateurs (clients et vendeurs) et leurs réputations sont fonction de la manière dont les achats et les ventes se sont déroulés sur le(s) site(s).

Une autre utilisation concerne les systèmes décentralisés, dans lesquels un système ne connaissant pas un client va contacter d'autres systèmes pour connaître sa réputation puis décider si celui-ci est digne ou non de confiance. Certains systèmes d'échanges utilisant le modèle *peer to peer* fonctionnent ainsi pour établir la priorité des échanges avec leurs clients.

Dans la section suivante, nous présentons quelques systèmes de gestion de confiance significatifs appliquant cette approche tels que modèle de Marsh [67], eBay [38], le modèle en logique subjective de Josang [55, 54], le modèle de réputation d'Abdul-Rahman et Hailes [2], le modèle de Shmatikov et Talcott[79] et le framework de Krukow *et al.* [59].

3.3.1 Modèle de Marsh

Marsh est le premier auteur qui a donné une formalisation du concept de confiance. Dans sa thèse [67], il propose un modèle général de *gestion de la confiance dans le cadre de la coopération* qui utilise la notion de réputation.

Il a défini trois sortes de réputations : *Basique, Générale et Situationnelle*. Ces trois réputations sont représentées par des valeurs dans $[-1, +1]$. Marsh estime que si la confiance aveugle existait, alors le fait de chercher à estimer la réputation d'une entité ne serait plus nécessaire. Il considère donc que la confiance aveugle n'existe pas et retire la valeur $+1$ du domaine des valeurs possibles pour les réputations. L'auteur ne précise pas d'où provient l'information qui permet de calculer les réputations, ni comment les réputations sont initialisées. Il s'est intéressé principalement aux processus de décision. Les réputations sont calculées par rapport à l'ensemble des interactions.

L'auteur a proposé pour ce modèle un processus de décision entièrement automatisé. Chaque décision est prise par seuillage : si la valeur de réputation *Situationnelle* est supérieure au seuil de coopération, alors l'entité coopère sinon elle ne coopère pas. Marsh a proposé également plusieurs définitions de ce seuil de coopération en fonction des risques, de la compétence de la cible et de l'importance de la situation perçus par le bénéficiaire.

Un modèle manipulant des réputations fondées sur des interactions directes entre cibles et bénéficiaires est proposé dans ce système. Ces réputation sont toutes subjectives, graduées et non transitives. La notion de dimension n'est ni présente ni déductible du modèle. Les processus de décision sont indépendants de l'humain.

3.3.2 eBay

Le site de vente aux enchères eBay[38] met en œuvre un schéma d'utilisation de réputation simple et représentatif. Ce système de réputation est basé sur l'historique des transactions passées de chaque acteur. Après chaque enchère, le vendeur et l'acheteur ont la possibilité de s'évaluer mutuellement en émettant un avis positif, neutre ou négatif sur la transaction. La réputation d'un utilisateur (acheteur ou marchand) est fonction de l'ensemble de ces avis, cette information est partagée dans le système.

Ce schéma de réputation est facilement compréhensible et est la base de la confiance des utilisateurs dans le système. Grace à l'ensemble des avis disponibles, le système peut

déterminer le pourcentage de transactions évaluées positivement par les utilisateurs. Cette information permettra de prendre la décision de réaliser ou non une vente ou un achat. Par contre, ce schéma de réputation est centralisé, toutes les informations concernant la réputation sont gérées par eBay qui connait donc les avis émis par chacun des utilisateurs.

Dans le cadre des sites de vente en ligne, il est possible d'utiliser un autre aspect de la réputation : l'appréciation sur les produits ou sur les marchands. Quand le client achève une transaction, il peut la noter sous la forme d'une appréciation (de 1 à 5 étoiles). L'ensemble de ces appréciations permet de disposer de la réputation des produits ou des vendeurs et les futurs clients peuvent consulter ces informations avant de se décider pour un achat. À présent, ce modèle est appliqué dans presque tous les sites de vente en ligne. Le site de vente Amazon en est un exemple typique [37].

3.3.3 Modèle d'Abdul-Rahman et Hailes

Abdul-Rahman et Hailes [2] ont proposé un modèle de calcul de confiance basé sur l'expérience et la recommandation dans le contexte spécifique d'une communauté ; il s'agit de réseau de type pair-à-pair.

La valeur de confiance est discrète et appartient à l'ensemble *{-2, -1, 0, +1, +2}* qui correspond respectivement à *{very untrustworthy, untrustworthy, neutral, trustworthy, verytrustworthy }*. Chaque entité de la communauté stocke ses valeurs de confiance dans les entités avec lesquelles elle a eu des contacts (son expérience). Les recommandations des autres entités au sujet de la confiance en une autre entité permettent d'enrichir leurs propres expériences sur cet individu. Ces valeurs de confiance sont calculées à partir de l'historique complet des interactions.

3.3.4 Modèle de A. Josang

Audun Josang a proposé une logique modale spécifique, dite logique subjective [50, 51, 55, 52] pour modéliser des systèmes de réputation. C'est une logique de croyance subjective de propositions incertaines. Le composant de base de cette logique est l'*opinion*. Une *opinion* est un quadruplet qui contient les éléments suivants : **b** - la croyance dans la proposition, **d** - l'incrédulité (disbelief) en la proposition, **u** - l'incertitude de la proposition et **a** - l'atomicité relative de la proposition. Ces valeurs sont de type numérique et sont lieés entre elles par la relation suivante : $b+d+u = 1$. La probabilité d'espérance est égale à $b + a * u$.

Le calcul de la confiance sur un contexte incertain est réalisé en raisonnant sur un chemin défini en terme de probabilité sur des événements incertains. Six opérateurs sont définis (*conjonction, disjonction, négation, consensus, actualisation (discounting) et inférence conditionnelle*) pour raisonner sur la confiance. Le système ne permet pas d'analyser et de faire des calculs sur la confiance si les chemins ne sont pas interconnectés.

L'opérateur *inférence conditionnelle* [49] permet de modéliser la situation où l'antécédent et la conséquence sont incertains, en lui donnant une valeur probabiliste. Supposons que l'affirmation x soit associée à l'opinion o_x et que l'hypothèse conditionnelle $x \rightarrow y$ soit associée à l'opinion $o_{x \rightarrow y}$. Ce qui nous intéresse dans ce cas, c'est de connaître l'opinion o_y sur la conclusion de y. C'est l'opérateur d'inférence conditionnelle qui traite cette opération.

3.3.5 Modèle de réputation de Shmatikov et Talcott

Ce modèle est proposé par Shmatikov et Talcott [79] et a été adapté par Krukow *et al.* [59] pour le framework qu'ils ont utilisé pour leur système de réputation. Ce modèle définit la notion de *licenses*. Une *license* formalise les restrictions et les obligations, elle explique ce que son propriétaire doit ou ne doit pas faire et ce qu'il peut ou ne peut pas faire. Une licence l est spécifiée par trois fonctions : l.**permits**, l.**violated** et l.**done**. L'historique **H** est l'ensemble des événements associés à un *principal*.

Les fonctions l.**violated** et l.**done** prennent **H** comme paramètre d'entrée et rendent une valeur booléenne. La fonction l.**violated** rend *true* si l'historique **H** viole la *license*. La fonction l.**done** rend *true* si la licence l a expiré dans l'historique **H**. La fonction l.**permits** prend deux paramètres en entrée : un événement e et l'historique H, elle rend *true* si et seulement si l'événement e peut être ajouté (autorisé) à la licence.

Comment un principal p qui disposerait d'une licence l et d'un historique H peut accéder à une ressource ? Le propriétaire de la ressource r effectue un contrôle des droits d'accès en prenant la licence l et l'historique H comme paramètre. Le droit d'accès à la ressource r est spécifié par son propriétaire en utilisant la méthode *r.useOK(l,H)*. La valeur de retour de cette méthode est *true* si le principal p peut accéder à la ressource avec sa licence l et son historique H. Dans le même temps, cet événement sera pris en compte et ajouté à l'historique H.

En utilisant les différents éléments définis ci-dessus : la licence l, les fonctions et la méthode de contrôle d'accès à la ressource $useOK$, un principal peut définir une politique qui donne ou non le droit d'accès à sa ressource r aux autres principals en fonction de leurs historiques.

3.3.6 Framework de réputation de Krukow *et al.*

Krukow *et al.* [59] ont proposé un framework qui utilise la logique de type LTL [74] pour formaliser des systèmes de réputation. Le framework se compose d'une structure d'événements, d'un langage de politique et d'un vérificateur de politique.

Dans ce framework, tous les comportements des entités sont observés en tant qu'événements. Lors d'une interaction, ces événements apparaissent dans un ordre qui est compatible avec la structure d'événements (structure qui décrit les relations entre les événements). Les événements relatifs à une interaction sont conservés sous la forme d'une séquence d'événements : une session. Les sessions sont sauvegardées dans l'*historique des interactions* de l'entité. Une logique modale (de type LTL [74]) qui s'applique aux faits ayant eu lieu dans le passé est mise en oeuvre comme langage d'expression de la politique. Avec ce langage, la politique de sécurité d'une entité est spécifiée sous la forme d'une formule logique. Le module de vérification a pour objectif de vérifier en permanence si un historique **H** satisfait ou pas une politique ψ : $\mathbf{H} \models \psi$. C'est cette valeur de vérité qui permet de déterminer si l'on peut faire confiance ou non.

Voici un bref exemple proposé par [59]. Nous nous plaçons dans le contexte d'un client qui veut effectuer une enchère sur le site eBay. Son comportement est modélisé sous la forme d'événements :

- `pay` : il paie pour la transaction quand le vendeur le lui demande,
- `ignore` : il ne paie pas et abandonne la transaction,
- `confirm` : il veut avoir une confirmation du vendeur après avoir réalisé le paiement,
- `time_out` : le vendeur a reçu le paiement et a abandonné la transaction,
- `positive/neutral/negative` : évaluation de la transaction avec le vendeur.

Le client peut avoir à un moment donné un historique **H** tel que :

$$\mathbf{H} = \texttt{\{pay,confirm,positive\}\{pay,confirm,neutral\}\{pay\}}$$

Le client dispose de la politique (représentée par une formule logique) :

$$\psi \equiv \neg F^{-1}(time\_out)$$

Cette politique indique que *le client ne participe qu'à des enchères proposées par les vendeurs qui ont toujours envoyé les produit payés*. La vérification $\mathbf{H} \models \psi$ fournit un résultat binaire *(oui/non)* qui indique au client s'il peut continuer ou non la transaction en cours. Une explication détaillée de ce scénario sera présentée dans le chapitre 4.

Ce framework permet de formaliser parfaitement les systèmes de réputation avec une syntaxe stricte. Les problèmes que posent ce framework sont le volume nécessaire au stockage de l'historique et l'algorithme de vérification de la politique qui est de complexité exponentielle (vérifier la satisfiabilité d'une formule de logique est un problème NP-complet).

3.4 Approches hybrides de la gestion de la confiance

Les approches de gestion de la confiance présentées précédemment, basées sur l'utilisation de politiques et sur l'utilisation de la réputation ne sont pas toujours efficace dans certaines situations réelles. Dans ces cas là, des approches *hybrides* doivent être prises en considération.

La confiance, dans l'approche basée sur l'usage de politiques, est mesurée directement par la politique qui contrôle les actions des acteurs du système. Par contre, dans le cas d'approches basées sur la réputation, la confiance est mesurée à partir des informations disponibles sur la réputation des acteurs. Les approches hybrides vont utiliser les propriétés de ces deux approches.

Nous présentons ici un scénario d'application qui a besoin de ce type d'approche *mixte* : un marchand (vendeur en-ligne) veut vérifier les commandes passées par ses clients avant de passer commande à ses propres fournisseurs. Le facteur le plus important pour valider une commande est lié aux risques de non-paiement des achats, que la somme soit grande (i.e, des centaines euros) ou qu'elle soit petite (i.e, quelques euros). Si le montant de l'achat est petit, on peut imaginer que le marchand a juste besoin de vérifier l'historique des achats de cet utilisateur (cet historique peut nous permettre de calculer sa réputation en utilisant le nombre d'achats évalués comme *positive*, par exemple) ; par contre si le montant est élevé, le marchand a besoin, par exemple, de vérifier la validité de la carte de crédit pour avoir une assurance forte d'être payé. Dans ce contexte, un modèle hybride est requis pour réaliser efficacement cette application.

Dans cette section, nous présentons le système SULTAN [36, 35] et le *framework* SECURE qui utilisent tous les deux cette approche.

3.4.1 SULTAN

SULTAN a été développé par Tyrone Grandion *et al.* [36, 35]. Il s'agit d'un système de gestion de la confiance qui fonctionne dans un contexte centralisé. L'architecture générale de SULTAN est présentée dans la figure 3.3.

Le système est composé des éléments suivants : le *State Information* qui contient toutes les informations sur les scénarios des applications (expériences et risques) ; le *Monitoring Service* qui observe et effectue la mise à jour des informations concernant le système ; le *Risk Service* qui identifie les risques à partir des modèles intégrés dans le

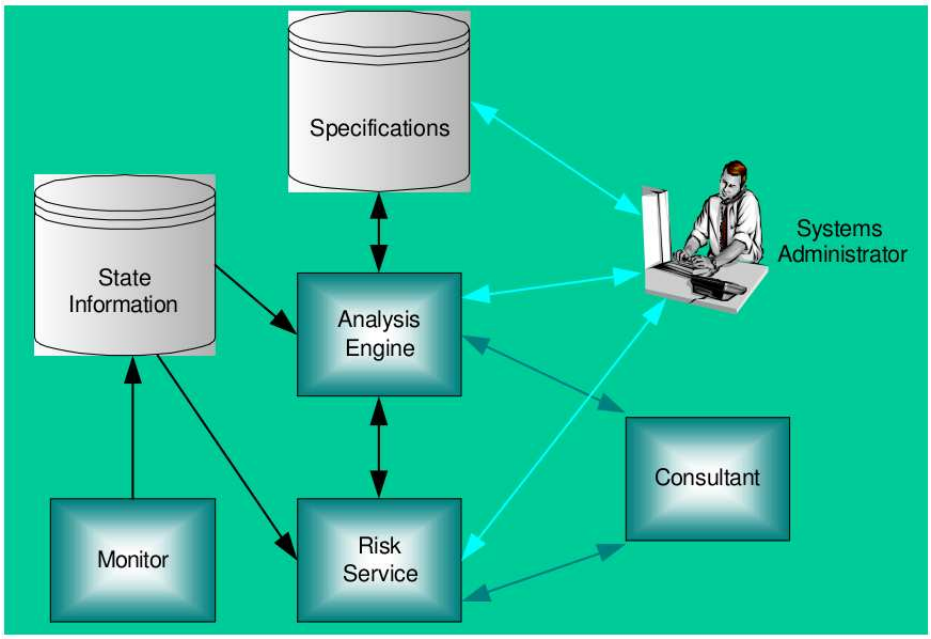


FIG. 3.3 – Framework SULTAN

système ; le *Specification Server* qui contient les prédicats qui spécifient la politique de sécurité ou les recommandations de chaque entité ; l'*Analysis Engine* qui est le moteur de raisonnement. Les raisonnements sont réalisés en utilisant l'ensemble des prédicats définis et le *niveau de risque* fourni par le *Risk Service*.

SULTAN dispose d'un langage formel, Ponder [70] qui utilise la logique des prédicats. Ce langage permet de spécifier les politiques de sécurité de chaque *entité* du système. Ces politiques se composent de plusieurs prédicats. Les prédicats expriment soit la confiance, soit des recommandations. L'exemple suivant est extrait de [35].

- Prédicat de confiance
  `name : trust (trustor, trustee, actions, level) ⊢ constraint set ;`
  Ce prédicat indique le niveau de confiance (`level`) du `trustor` envers le `trustee` pour l'action `actions` lorsque l'ensemble des contraintes *contraints* est satisfait.
  Exemple 1 : `trust (Morris, doctor, heart_diagnosis(Morris) : operate(Morris), 50) ⊢ is_consultant(_doctor, NHLI) ;`
- Prédicat de recommandation
  `name : recommend (recommendor, recommendee, actions, level) ⊢ constraint set ;`
  Ce prédicat indique le niveau de recommandation (`level`) de `recommendor` en *recommendee* pour l'action *actions* lorsque l'ensemble des contraintes *contraints* est satisfait.
  Exemple 2 : `recommend (Tyrone, joePublic, WebProgram(joePublic), high) ⊢ has_degree(joePublic, IC_Computing, 2i) ;`

3.4.2 Framework SECURE

SECURE (Secure Environments for Collaboration among Ubiquitous Roaming Entities) est un *framework* de gestion de la confiance développé dans le cadre du projet européen éponyme SECURE [15, 14, 16]. Le composant important du framework est un modèle de confiance qui fournit les bases pour représenter la confiance, raisonner sur la confiance et représenter les politiques de sécurité basées sur la confiance.

L'application comporte plusieurs acteurs, des principals selon la terminologie SECURE, qui ont des interactions entre eux. Chaque principal p de l'application dispose d'un module de confiance, utilisé quand l'application doit prendre une décision lors d'une interaction avec un autre principal. Chaque module de confiance se compose de trois composants : le moteur de confiance, le moteur de risque et l'outil de collaboration. L'outil de collaboration sauvegarde tous les comportements des principals avec lequels p a des interactions. Cette information est utilisée avec la politique de confiance de p, cela permet de connaitre le niveau de confiance que p a en chaque principal du système. Dans ce modèle, la confiance est un élément utilisé pour l'analyse du risque. Le module de confiance, avec différents composants est illustré par la figure 3.4

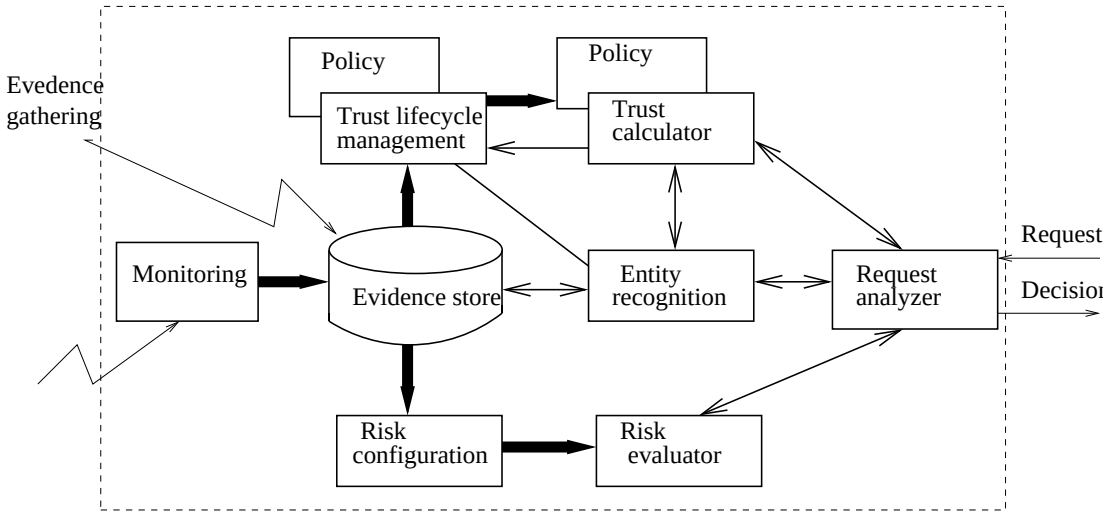


FIG. 3.4 – SECURE framework

Dans la conception du framework, les informations liées à l'évaluation du risque et de la confiance (l'expérience, les comportements dans le passé) sont prises en compte lors des décisions. La valeur de confiance attribuée à un principal est évaluée en utilisant les expériences antérieures avec ce principal et les recommandations des autres principals le concernant. La politique est définie comme la confiance d'un principal dans les autres principals. Le modèle du risque proposé dans SECURE est un modèle de calcul du risqué basé sur les conséquences et leurs probabilités. Le détail sur modèle du risque de SECURE est donné dans la section de la gestion du risque.

Au niveau formel, le framework est décrit avec les types abstrait du $\lambda - calcul$ [82]. Avec cette formalisation, les élements de base du système tels que la confiance, la politique de confiance de chaque principal ou la politique global du système sont représentés clairement. Nous reviendrons sur cette représentation de SECURE en décrivant notre approche pour un système de gestion de la confiance dans le chapitre 4.

En conlusion, le framework SECURE est un modèle de confiance riche de par sa conception. Dans ce modèle, la confiance est mesurée par plusieurs sources d'informations telles que la réputation et les recommandations. Le modèle permet de proposer à un acteur son propre mécanisme pour calculer la confiance et le risque. Par contre, ce framework n'est qu'un squelette de développement, pour pouvoir l'utiliser, il faudrait disposer de composants supplémentaires tels qu'un langage d'expression des politiques, un mécanisme d'interprétation du risque et une représentation opérationnelle de la confiance.

3.5 Système de négociation de la confiance

Le sytème de *négociation de la confiance* désigne un système de gestion de la confiance qui permet aux deux parties impliquées d'établir une relation de confiance mutuelle. Les deux parties sont considérées de façon identique (comme ayant le même statut) dans l'établissement de la relation et ce type de mécanisme est parfaitement adapté à l'établissement de la confiance dans les réseaux pair à pair.

Dans cette approche, la confiance entre les deux entités est établie de façon incrémentale par l'échange d'informations étape par étape. Cette façon de procéder distingue la négociation de la confiance des systèmes traditionnels de gestion de confiance où la confiance est établie en une seule étape par la vérification de la politique et des qualifications fournies.

Nous pouvons considérer que la négociation de la confiance est un protocole et qu'il peut être mis en œuvre de façon automatique ou interactive. De ce point de vue, la négociation est alors une succession d'échanges et de vérifications des éléments de preuve fournis par chacune des parties. Dans les systèmes de négociation que nous présenterons par la suite, les éléments de preuve échangés sont des qualifications.

Tout comme un système traditionnel de gestion de la confiance, un système de négociation de confiance se doit de disposer d'un langage pour spécifier les politiques et les qualifications de chaque partie et un module de vérification pour s'assurer de la conformité de ces politiques et de ces qualifications. En plus de ces composants, un système de négociation de la confiance doit disposer d'un mécanisme qui permette la mise en œuvre de la *stratégie de négociation* de chacun des participants.

Nous étudierons dans ce chapitre les systèmes de négociation considérés comme les plus significatifs dans la littérature : Trust-X [5, 6], PeerTrust [85, 19, 77] et PROTUNE [13, 73].

3.5.1 Trust-X

Le système de négociation de confiance Trust-X [5, 6] a été développé par E. Bertino *et al.* Il fournit un langage appelé X-TNL (basé sur le formalisme de XML) [22] pour spécifier les politiques et les qualifications. L'architecture de Trust-X est présentée dans la figure 3.5.

La négociation a lieu entre deux entités : celle qui contrôle la ressource (le contrôleur) et celle qui la demande (le demandeur). Chacune des entités est caractérisée par son profil *Trust-X* qui indique quel est son rôle dans la transaction. Le processus de négociation pour établir une relation de confiance mutuelle entre deux parties est le suivant : le demandeur envoie ses qualifications pour accéder à la ressource, le contrôleur lui envoie son certificat pour établir une communication sécurisée, puis à chaque étape, des informations sensibles sont échangées dans le respect de la politique de divulgation

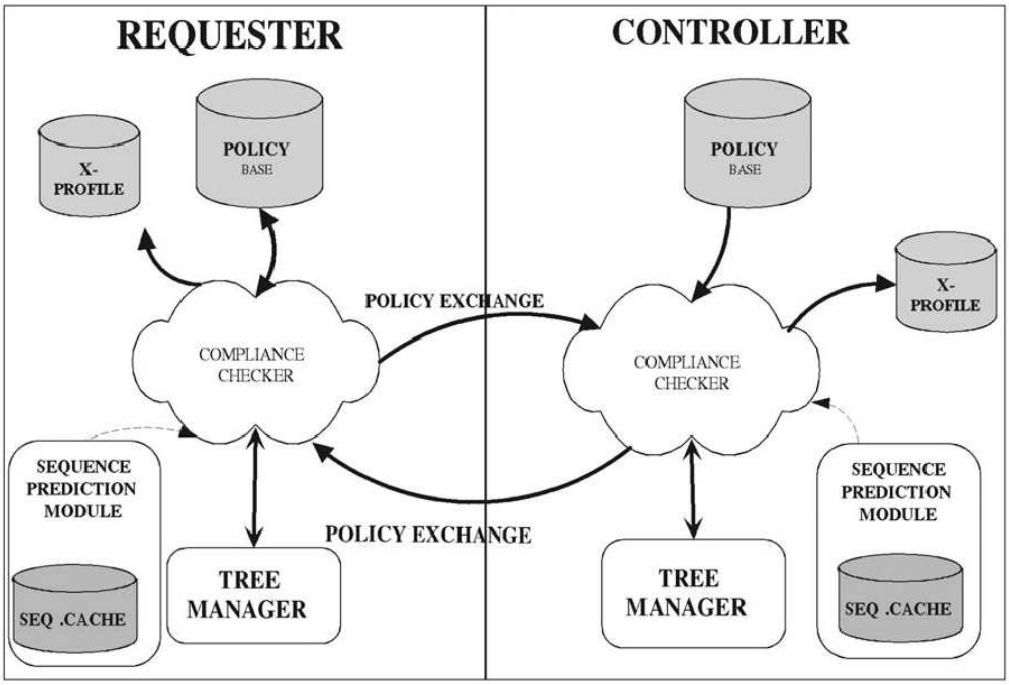


FIG. 3.5 – Architecture du système de négociation Trust-X

de chacun. La négociation s'achève lorsque les échanges et le raisonnement associé à la politique et aux qualifications se terminent.

Comme illustré dans la figure 3.5, un module de négociation est associé à chaque pair présent dans le système. Chaque module dispose des composants suivants : une base de la politique (Policy Base) qui gère des politique d'échanges ; un gestionnaire des états de négociation (Tree Mangager) et un vérificateur de satisfaction de la politique (Compliance Checker).

Les données échangées dans la négociation sont des *certificats*. Un *certificat* dans ce système peut être une qualification (credential) ou une déclaration. Elles sont encodées dans le langage X-TNL. Une *qualification* est un ensemble de propriétés signées par une autorité de certifcation reconue. Ces informations au format XML sont signées en respectant la norme *Xml signature* du W3C [81]. Une *déclaration* est, par contre, un ensemble de données sans certification (elles n'engagent que leur porteur).

Voici un exemple de qualification et de déclaration rédigées en X-TNL (extrait de [6]) :

– Qualification

```
<Corrier_Employee credID='12ab', SENS='NORMAL'>
    <Issuer HREF='http://www.corrier.com' Title=Corrier_Employee_Resposity/>
    <name>
        <Fname> Olivia </Fname>
    </name>
    <adress> Grange Wood 69 Dublin </adress>
    <employee_number code='34ABN'/>
    <position> driver </position>
</Corrier_Employee>
```

Cet exemple illustre une qualification contenant les informations personnelles d'un employé de la société *Corrier*. Elle est signée et attribuée par la société *Corrier* (`<Issuer HREF='http ://www.corrier.com'>`) qui se comporte en autorité de certification dans cette situation. La partie *XML signature* qui doit être associée à cette qualification n'est pas représentée dans cet exemple.

– Déclaration :

```
<car_preferences>
    <name>
    <Fname> Olivia </Fname>
    <Fname> White </Fname>
    </name>
    <car_category> compact </car_category>
    <vehicule>
        <model>TOYOTA CORROLA 1.4</model>
        <model>FIAT PUNTO 1.2</model>
    </vehicule>
</car_preferences>
```

Cette déclaration décrit les préférences d'Olivia en terme de voiture. Cette information peut être utilisée lors la négociation entre un *employé* de *Corrier* avec une société de location de véhicules.

La politique est basée sur l'usage de certificats et utilise les notions de *R-Term*, de *Policy condition*, de *Term* et de *Disclosure Policy* qui sont définis de la manière suivante :

– *R-Term* : une expression de la forme *ressource_name (attribute_list)* où *ressource_name* identifie une ressource ou un service et où *attribute_list* est une liste d'attributs spécifiés par la ressource.
– *Policy condition* : une expression logique de la forme : *a op expr* où *a* est une propriété du certificat auquel elle est appliquée ; *op* est un opérateur de comparaison tels que $\neq, <, >, =, \leq$ et *expr* est une constante ou une expression d'un type compatible avec *a*.
– *Term* : un terme T est défini sous les formes suivantes :

1. $P(C)$ où P est un certificat et C est une liste de *Policy conditions* (éventuellement vide) appliquée à P ;

2. $X(C)$ où X est une variable et C est une liste de *Policy conditions* non vide. Cette seconde forme permet d'imposer des conditions sans avoir à spécifier a-priori à qui elles doivent être appliquées.

– *Disclosure policy* : soit R une ressource, une *Disclosure policy* p pour R est une paire *(pol_prec_set, rule)* où *pol_prec_set* est un ensemble de politiques préalablement définies et *rule* est une expression sous une des formes suivantes :

1. $R \leftarrow T_1, \ldots T_n$ où R est le nom de la ressource et les T_i des *Term* ;

2. $R \leftarrow DELIV$ qui indique alors que la ressource R peut être divulguée sans condition.

Nous fournissons ici un exemple de politique extrait de [6] :

$$\begin{aligned} pol_1 = &(\{\} Rental\_Car \leftarrow \\ &Corrier\_Employee(code = Rental\_Car.request\_Code, position = driver), \\ &Id\_Card(name = Corrier\_Employee.name)) \end{aligned}$$

Cette politique explique que le service *Rental_ Car* est accessible aux employés de la société *Corrier*. Cette règle doit être prouvée avec les qualifications *Corrier_ Employee* et *Id_ Card*.

Le processus de négociation se déroule en trois phases : *introduction*, *génération de séquences* et *échange de certificats*. La phase *introduction* permet de spécifier l'objet de la négociation (ressource/service) et d'échanger les politiques de divulgation des entités. La phrase *génération de séquences* permet de déterminer quelles sont les informations à échanger tout en respectant les politiques de chacun. La phrase *échange de certificats* permet d'échanger et de vérifier les qualifications déterminées comme nécessaires lors de la phase précédente.

3.5.2 PeerTrust

Le système de négociation PeerTrust [85, 19, 77] a été développé dans le contexte d'un accès sécurisé et de confiance au web sémantique. Ce système s'appuie sur des politiques basées sur le contrôle d'accès.

L'objectif de la négociation est de déterminer une séquence de qualifications $(C_1, \ldots C_i, R)$ ou R est la ressource à laquelle on veut accéder. La construction de cette séquence est réalisée de la manière suivante : R est est accessible si C_i est présentée, C_i est présentée (accessible) si on a pu déterminer la séquence de qualifications $(C_1, \ldots C_{i-1}, C_i)$ et ainsi de suite jusqu'à ce que plus aucune qualification ne soit nécessaire.

Le système fournit un langage de politique qui s'exprime sous la forme de la logique du premier ordre (clause de Horn [41]). Les règles sont de la forme :

$$lit \leftarrow lit_1, \ldots, lit_n$$

Chaque lit_i est un littéral positif $P_j(t_1, \ldots, t_n)$ où P_j est un prédicat et les t_i sont les arguments de ce prédicat. Chaque t_i est un *terme* qui peut être une fonction avec ses arguments.

Pour faire référence aux autres entités dans le système, PeerTrust utilise un mécanisme de raisonnement sur les déclarations de ces entités. Pour exprimer la délégation et l'évaluation à une autre entité, il utilise la règle où les littéraux lit_i contiennent l'argument *Authority*.

$$lit_i@Authority$$

où *Authority* désigne l'entité qui a la responsabilité ou le droit d'évaluer le terme lit_i. L'argument *Authority* peut être imbriqué. Le raisonnement sur ces règles contenant les paramètres de l'autorisation de l'entité permet de donner un chemin d'établissement de la confiance entre les entités et c'est le principe du système.

Une implémentation de PeerTrust 1.0 a été réalisée pour démontrer son fonctionnement. Le cœur du système se compose des règles de la politique et de son moteur de raisonnement développé en Prolog. Dans le système, un module de négociation, le *trust agent*, est associé à chaque entité. L'architecture d'un *trust agent* est illustrée dans la figure 3.6. Dans ce module, *Interface* assure les communications avec les autres

agents ; *Credential Verification* vérifie la validité des qualifications présentées ; *Inference engine* s'occupe de raisonner sur les prédicats, les qualifications, les politiques... Les composants *Strategy evaluator* et *Negotiation* implémentent la stratégie du processus de négociation.

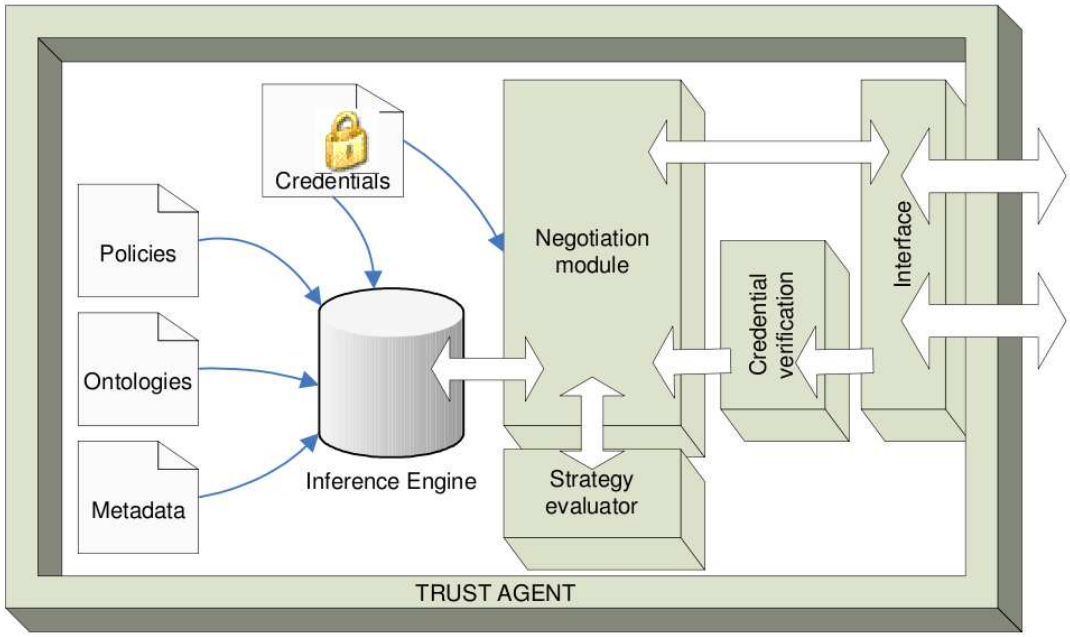


FIG. 3.6 – Architecture de PeerTrust

3.5.3 PROTUNE

Le framework PROTUNE [72, 13] a été développé pour permettre la négociation entre services Web. Le composant le plus important de ce framework est le langage de description des politiques, proche de Ponder [70]. Ce langage est très riche et permet d'exprimer un modèle complet ou la confiance est mesurée par toutes sortes d'informations : les credentials, la réputation, la recommandation...

Les actions, la politique, les credentials et la confiance sont présentés sous la forme de prédicats logiques. Chaque acteur a une base de connaissances qui contient les prédicats exprimant sa politique, le niveau de confiance dans les autres acteurs, le niveau du risque acceptable pour chaque action... Les informations sur la confiance et le risque sont évaluées à partir d'une base de données contenant l'historique des interactions passées.

Le processus de négociation est réalisé par un moteur d'analyse qui raisonne sur les prédicats impliquant les actions requises, les politiques... L'architecture générale de PROTUNE est présentée dans la figure 3.7, extrait de [72].

Dans cette architecture, le client peut envoyer plusieurs requêtes successives au serveur pour négocier le meilleur service possible. Les requêtes sont spécifiées en utilisant le langage de PROTUNE et elles sont de l'un des types suivant :

- *Acces Control* : les requêtes concernant le droit d'accès au service ou à la ressource ;
- *Why* : lorsqu'une décision a été prise lors de la négociation, le client peut demander au serveur des explications (pourquoi a-t-on obtenu ou non la ressource, par exemple) ;

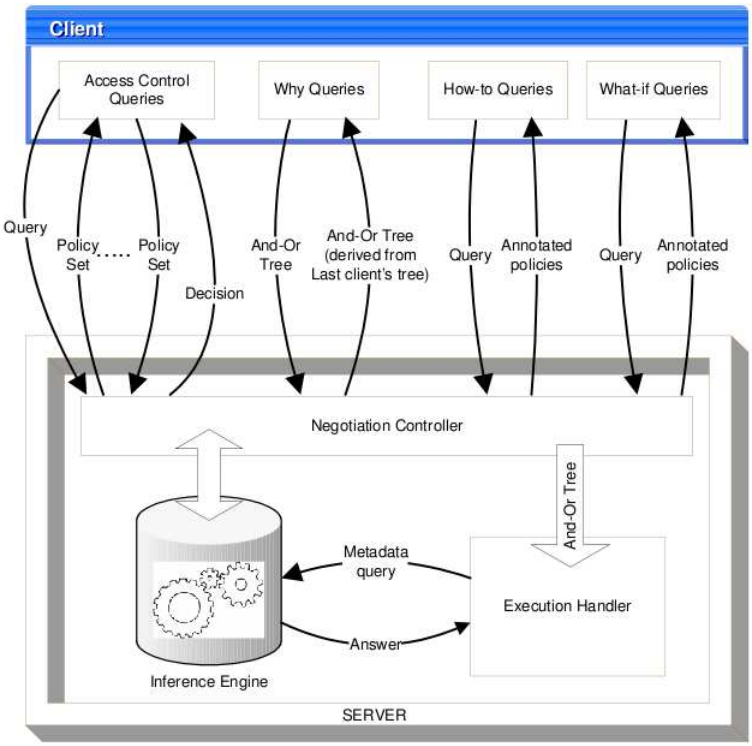


FIG. 3.7 – Architecture de Protune

- *Howto* : le client demande au serveur quelles sont les conditions qu'il doit satisfaire pour pouvoir accéder à la ressource ou au service ;
- *WhatIf* : le client propose au serveur une situation hypothétique où il veut obtenir la ressource ou le service.

Le serveur se compose de trois modules : *Negotiation Controller*, *Inference Engine* et *Execution Handler*. Le *Negotiation Controller* est l'interface d'interaction auprès du client et contrôle le processus de négociation. Le *Inference Engine* contient les politiques et réalise le raisonnement sur ces politiques pour une requête donnée. Le *Execution Handler* est en charge d'exécution des actions sur le système.

3.6 Prise en charge du risque pour la gestion de confiance

Le risque est un facteur très important qui concerne strictement la gestion de la confiance. Lorsque l'on évolue dans un environnement sûr, dans lequel nous n'avons pas identifié de risques, il n'est pas nécessaire de faire appel à la confiance. Par contre, lorsque l'on évolue dans un environnement incertain, dans lequel à chaque action entreprise sont attachés certains risques, les mécanismes de gestion de la confiance prennent tout leur intérêt. Le problème qui se pose est donc : comment peut-on modéliser les risques et les prendre en compte dans le cas des prises de décision reposant sur la notion de confiance ? Pour les applications informatiques, le risque peut être abordé de deux manières différentes : par une approche qualitative ou par une approche quantitative.

3.6.1 Approche qualitative

Comme nous l'avons vu au chapitre 2, le risque est défini comme étant une combinaison de la probabilité d'un événement et de ses conséquences. Dans cette aproche, nous n'estimons que la perte potentielle liée à chaque conséquence et ne prenons pas en compte sa probabilité. Nous utilisons trois facteurs de base pour évaluer les conséquences : *les menaces*, les *vulnérabilités* et les *contrôles*. Le risque sera une évaluation des conséquences basée sur ces facteurs.

- *Les menaces* (threats) : incluent tout ce qui est extérieur au système. Elles en sont indépendantes et peuvent survenir de manière imprévue. Les *incendies*, les *fraudes* ou les *pannes* sont des exemples de menaces.
- *Les vulnérabilités* sont des éléments connus du système qui augmentent le coût des conséquences lorsque survient une menace. Par exemple, la présence de produit inflammable augmentera l'impact d'un incendie.
- *Les contrôles* (controls) sont les contre-mesures qui permettent de réduire le potentiel des vulnérabilités. Les principales contre-mesures appartiennent aux catégories suivantes : la dissuasion, la prévention, la correction et la détection.

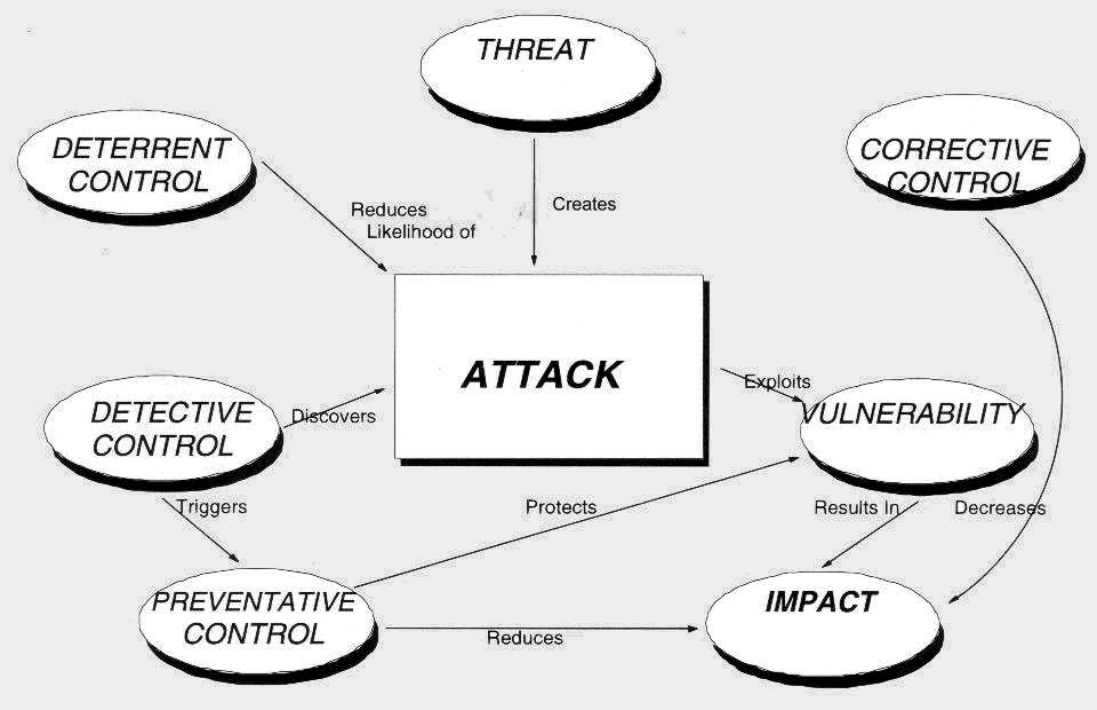


Fig. 3.8 – Risque qualitatif

En identifiant ces facteurs et en analysant les interactions, nous pouvons en estimer l'impact sur le système. Ensuite, nous pouvons proposer de mettre en place des mesures efficaces pour en réduire les impacts. La figure 3.8, extraite de [1] présente les différentes interactions entre les facteurs de ce modèle de risque.

3.6.2 Approche quantitative

Pour cette aproche, on considère que chaque action ou transaction peut avoir une ou plusieurs conséquences et que chacune de ces conséquences a un coût et une proba-

bilité de se réaliser. Pour chaque action, il est en général possible de modéliser et de calculer une valeur de risque. En général, cette valeur est calculée comme le produit de la probabilité d'occurrence d'une conséquence et de son coût. Si nous supposons que la probabilité d'occurrence de chacune des conséquences E est $p(E)$ et le coût en est $c(E)$, le risque serait $\Sigma(p(E) * c(E))$.

L'idée de cette approche du risque est simple mais il y a certains problèmes que l'on doit prendre en compte :

- Il faut trouver une bonne interprétation des couples *(probabilité, coût)*.
- Il faut trouver une méthode pour considérer toutes les conséquences de l'action.
- Il faut proposer une formule de calcul du risque satisfaisante pour chaque contexte d'application.

Une fois que le risque est déterminé, sa combinaison avec l'évaluation de la confiance nous permet de de prendre efficacement notre décision. Certaines infrastructures de gestion de la confiance, tel que SULTAN [28, 36] et SECURE [15], ont pris en charge la notion de risque dans leurs mécanismes de décision.

3.6.3 Modèles du risque

Nous présentons dans cette section deux modèles de risque que nous considérons comme significatifs et qui sont utilisés dans les systèmes de gestion de la confiance que nous avons présenté précédemment : le modèle de risque de SULTAN [28, 36] et celui de SECURE [15].

Modèle de risque de SULTAN

Ce modèle incorpore les aspects qualitatifs et quantitatifs du risque. Il est appliqué aux transactions Internet. Le modèle évalue le risque pour une transaction particulière en combinant les différentes informations dont dispose le *risk service* (informations concernant le risque lié aux transactions précédentes). Le résulat de cette évaluation sert à prendre les décisions qui concernent la confiance. La figure 3.9 illustre le modèle de risque utilisé dans SULTAN. Ce modèle propose une solution aux différents aspects de la modélisation du risque :

- *Détermination des risques et de leurs probabilités* : les catégories de risques liées à la transaction sont identifiées ; par exemple le *déni de service*, le *non paiement* ou *transaction illégale*. Chaque risque est identifié par un *id* et une probabilité p.
- *Détermination des pertes potentielles* : calcul qui se base sur la valeur des ressources de la transaction. S'il y a plusieurs ressources engagées dans la transaction, la valeur totale est estimée avec la contribution possible de chaque ressource.
- *Gestion des dépendances* : elle permet de déterminer la dépendance entre les actions et de déterminer si une conséquence finale identifiée comme étant un risque. Les probablités conditionnelles des actions sont calculées avec le théorème de Bayès. Dans le modèle, une méthode basée sur la logique subjective [55, 53] est proposée pour ce calcul de dépendance.
- *Gestion des profils de risque* : il s'agit de l'échantillonnage des niveaux de risque utilisés dans le raisonnment, le but est de les évaluer en fonction des différents seuils.
- *Calcul du risque* : il combine les aspects précédents. Il identifie le risque et sa probabilité, il calcule les pertes potentielles et en déduit la valeur du risque. Cette valeur est sauvegardée dans le *risk service*.

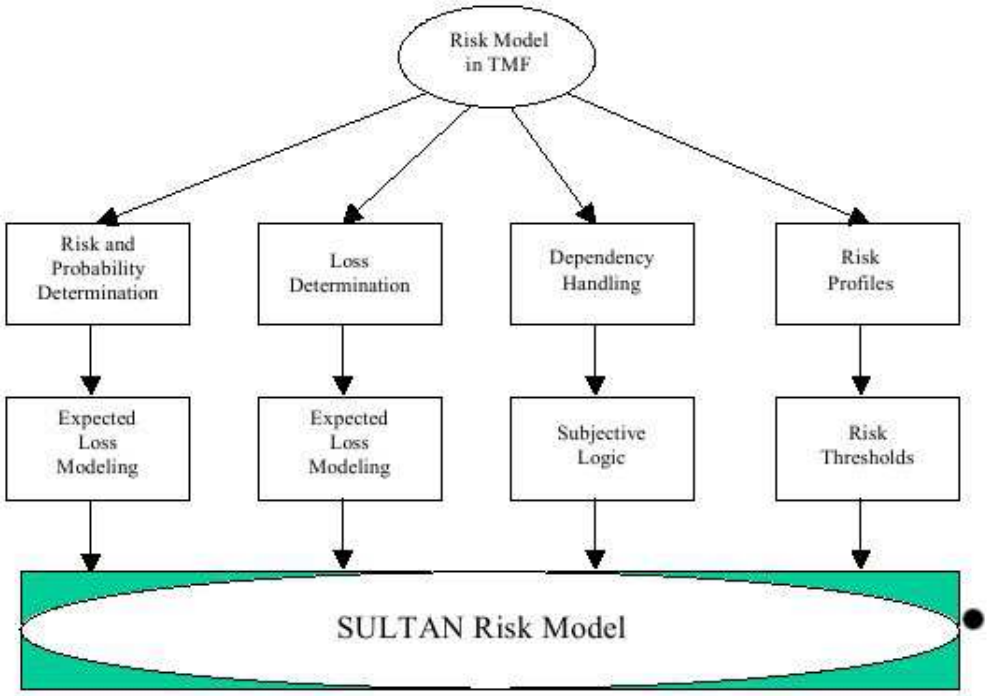


FIG. 3.9 – Modèle de risque de SULTAN

Modèle de risque de SECURE

Le risque dans ce modèle est évalué par l'estimation du *coût/bénéfice* des conséquences de chaque action, le modèle est illustré dans la figure 3.10.

Une action a du principal p peut avoir plusieurs conséquences (des résultats possibles différents évalués par leurs coûts/bénéfices). Avant d'entreprendre l'action a, les valeurs de coût/bénéfice potentielles de chaque conséquence sont estimées. Ces facteurs sont évalués par une famille de fonctions particulières que les auteurs du projet appellent des *cost-PDFs* (Cost-Probability Density Function).

Un autre point important de ce modèle de risque est qu'il prend en compte des informations sur la confiance relatives à l'action comme paramètre pour l'évaluation du risque. Cette valeur est utilisée comme paramètre la fonction cost-PDFs. Une fois que la valeur des *cost-PDF* de chaque conséquence est déterminée, elles sont combinées ensemble en respectant la politique de sécurité de chaque principal et permettent alors de prendre les décisions nécessaires.

3.7 Analyse des approches de gestion de confiance

Nous avons vu que les systèmes de gestion de la confiance décrits dans la littérature sont majoritairement développés en utilisant trois approches différentes : *l'approche basée sur la politique*, *l'approche basée sur l'expérience* et *l'approche hybride*. Le choix de l'approche dépend du contexte et du domaine de l'application qui est visée.

Les approches basées sur les politiques sont adaptées aux applications qui doivent respecter de fortes contraintes en termes d'autorisations et d'authentifications. Les systèmes de gestion de la confiance développés utilisent soit des mécanismes de raisonne-

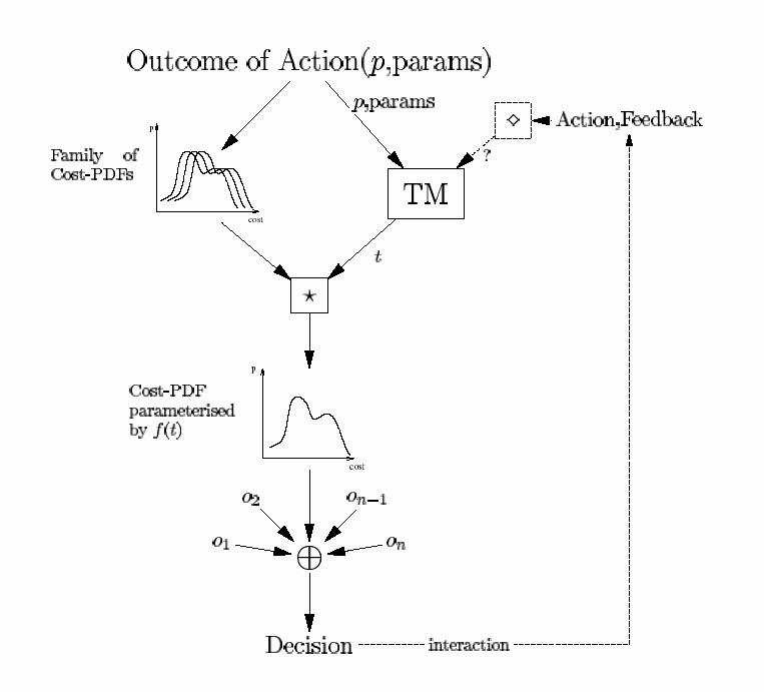


FIG. 3.10 – Modèle du risque SECURE

ment (RT, SD3) soit de la sécurité renforcée (KeyNote, TrustBuilder) pour modéliser et évaluer les relations de confiance. Le résultat fourni par ces systèmes est une décision binaire (*oui/non*) qui indique si l'entité fait ou ne fait pas confiance à l'autre entité pour autoriser l'action ou le service qui est demandé.

Les approches basées sur l'expérience sont utilisées en général lorsque les environnements sont moins structurés, avec des relations entre les entités plus souples et moins contraignantes. Pour ces approches, nous pouvons distinguer deux axes dans la conception des systèmes. Le premier axe utilise une *méthode de calcul* qui prend en compte les données concernant l'historique des participants pour déterminer la confiance qui peut être accordée, comme par exemple dans eBay, le modèle de Marsh ou celui de Abdul-Rahman et Hailes.

Dans le deuxième axe de ces approches, les décisions sur la confiance sont prises en effectuant des raisonnements sur les historiques, la réputation... des participants. Les modèles de Shmatikov et Talcott, de Josang et celui de Krukow en sont de bons exemples.

Les approches *hybrides* sont également utilisées pour développer des applications qui fonctionnent dans des environnements pair-à-pair et lorsque ces systèmes de confiance doivent être plus flexibles. Ces approches utilisent à la fois les techniques de raisonnement et des calculs pour évaluer la confiance (SULTAN). Cette approche semble permettre de concevoir des systèmes plus complexes en pratique.

Les différentes approches que nous avons présentées dans les sections précédentes montrent qu'il subsiste encore de nombreux problèmes dans les systèmes de gestion de confiance : soit le langage de politique n'est pas suffisamment riche pour exprimer complètement la confiance (KeyNote, TrustBuilder, SECURE), soit il manque un mécanisme de négociation au modèle (KeyNote), soit l'aspect centralisé du système le

rend difficile, voire impossible à mettre en œuvre dans des environnements trop vastes comme, par exemple, l'Internet (SULTAN, PROTUNE).

Nous allons détailler cette analyse par une comparaison de ces différents systèmes dans la section suivante.

3.8 Comparaison des systèmes

Dans ce chapitre, nous avons décrit les différents systèmes de gestion de la confiance. Afin de définir une infrastructure de gestion de la confiance qui corresponde aux objectifs que nous avons exprimés en introduction, nous allons maintenant revenir sur ces différents modèles de gestion de confiance et analyser quelques critères fondamentaux qui ont été utilisés pour chacun d'entre eux. Et à partir de cela, nous pouvons proposer des contraintes précises pour développer notre propre infrastructure de confiance.

Les systèmes de gestion de la confiance tels qu'ils sont présentés aujourd'hui dans la littérature se composent des éléments suivants :

- un langage d'expression de la politique : la politique définit les contraintes sur le droit d'accès aux services, le seuil de décision... Elle est spécifiée dans ce langage. Dans les systèmes comme KeyNote [10, 8] ou TrustBuilder [27, 25], ce langage permet aussi de spécifier les *credentials* échangés entre les entités lors du processus d'établissement de la confiance. Les actions que peut entreprendre une entité pourraient être spécifiées dans ce langage de manière à pouvoir les utiliser par la suite.
- Un module de prise de décision : il permet de raisonner ou de calculer la confiance en utilisant l'ensemble des données du modèle. Si le système est basé sur l'utilisation d'une politique, les données en entrée se composent de la politique de l'entité, des différents *credentials* échangés et de l'action entreprise. Le module raisonne sur cet ensemble de données pour fournir sa décision sur la confiance.
  Si le système est basé sur la réputation, les données sont composées de recommandations des autres entités, de la spécification de l'action et, selon les modèles, de l'historique des transactions de cette entité. Le module calculera une valeur de confiance pour l'entité à qui est demandée l'action.

Les systèmes de gestion de confiance sont dévéloppés pour des applications qui sont utilisées dans des contextes décentralisés ou pair-à-pair. Comme nous venons de le voir dans les différents systèmes étudiés, la centralisation de la gestion de la confiance est un point qui peut poser de nombreux problèmes pour ces applications, nous citons au moins :

- le système qui les héberge devient incontournable ;
- ses intérêts peuvent diverger de ceux de ses utilisateurs ;
- tout repose sur l'honnêteté du service ;
- il dispose d'informations très privilégiées ;
- respect des données personnelles.

Le premier point auquel nous devons nous intéresser dans la conception des mécanismes de gestion de la confiance est la prise en compte de la décentralisation de l'expérience, des credentials, des politiques et de la prise de décision. L'expérience sur les transactions passées doit être partagée et distribuée entre les entités de la communauté. Chaque entité doit gérer sa propre politique. Les décisions sur la confiance doivent être prises localement par chaque entité.

La plupart des frameworks présentés dans le chapitre précédent tel que KeyNote [10, 8] TrustBuilder [27, 25] sont décentralisés. Dans Keynote, chaque entité gère sa

propre politique grâce aux credentials qui lui ont été remis et aux assertions qu'il définit. TrustBuilder permet aussi à chaque entité de définir sa propre politique par des formules logiques. Par contre, le framework SULTAN[35, 36] est centralisé : toutes les *preuves* sont stockées dans une base commune. La politique est définie pour toutes les entités participant au système et les raisonnements sur la confiance et le risque sont réalisés de manière centralisée sur la base de connaissances commune.

Le langage de description de politiques est l'élément le plus important des infrastructures de gestion de la confiance. Comme nous l'avons indiqué au-dessus, il nous permet de représenter la notion de confiance avec son modèle, de spécifier la politique et les actions de chaque acteur... Chaque système de gestion de la confiance que nous avons étudié ci-dessus a son propre langage de politique. Nous présentons dans le tableau suivant une comparaison entre ces différents langages d'expression de politiques.

Le langage de politique doit satisfaire quelques critères tels que : une sémantique bien définie, la possibilité de présenter et de raisonner sur la confiance.... Nous considérons les critères suivants pour distinguer la richesse des langages de politique utilisés dans les systèmes de gestion de confiance présentés :

- La sémantique du langage est-elle bien définie (Y/N) ?
- Quel est le modèle de confiance choisi : modèle basé sur les *credentials* (C), la réputation (R) ou mixte (M) ?
- Le modèle permet-il de représenter des conditions complexes (Y/N) ?
- Le modèle supporte-t-il un mécanisme de protection des données sensibles de la politique (Y/N) ?
- Le modèle offre-t-il la possibilité de calculer ou de raisonner sur la confiance (Y/N) ?
- Le langage a-t-il la capacité d'exprimer la notion de risque pour un modèle de confiance complexe (Y/N) ?

Le tableau suivant nous présente une vue synthétique des différents langages utilisés dans les systèmes que nous avons étudiés.

Langage	KeyNote	Ponder	Protune	TPL
Sémantique bien définie	Y	Y	Y	Y
Modèle de confiance	C	M	M	C
Support des conditions complexes	Y	Y	Y	Y
Protection des données sensibles	N	N	N	N
Raisonnement sur la confiance	Y	Y	Y	Y
Prise en charge du risque	N	Y	Y	N
Appliqué dans	KeyNote	Sultan	Protune	Trustbuider

3.9 Synthèse

Dans ce chapitre, nous avons présenté les différentes approches des systèmes de gestion de la confiance que l'on trouve dans la littérature : l'approche basée sur la politique, l'approche basée sur l'expérience et l'approche hybride. Pour chaque approche, nous avons présenté quelques systèmes de gestion de confiance significatifs. À la fin de ce chapitre, nous proposons une analyse de ces systèmes de confiance et nous en précisons les points forts et les points faibles. À partir de cette analyse, nous présentons les contraintes que nous nous fixons pour la conception de notre infrastructure de gestion de la confiance. Il s'agira d'un système décentralisé qui permettra d'utiliser toutes les sources d'informations disponibles pour la prise de décision sur la confiance.

Chapitre 4

Vers un système de gestion de la confiance

Dans ce chapitre, nous présentons les principes de notre système de gestion de la confiance. Il s'agit d'un modèle destiné à un usage général, qui utilise toutes les sources d'informations à sa disposition pour évaluer la confiance. Nous présentons d'abord les propriétés que doit posséder notre système et nous les comparons aux systèmes existants. Ensuite, nous présentons les éléments fondamentaux qui nous permettent de construire notre système : il s'agit de la logique modale PP-LTL (Pure-Past Linear Temporal Logic) et du modèle de structure d'événements.

Plusieurs systèmes de gestion de la confiance proposés dans la littérature ont été présentés et analysés au chapitre 3. Nous avons vu les points forts et les points faibles de chacun d'entre eux. Le contenu de ce chapitre va nous servir de base à la conception de notre propre système de gestion de la confiance. Nous précisons également que celui-ci se classe dans la catégorie des *approches hybrides*.

4.1 Contraintes pour un système de gestion de la confiance

Nous voulons que le système de gestion de la confiance que nous proposons soit utile à la construction d'applications dans l'environnement d'Internet. Il s'agit d'applications qui doivent fonctionner dans un environnement distribué et dans des contextes de structuration de la communauté d'usagers différents [13] (communautés structurées, environnement pair-à-pair...).

Par environnement structuré nous entendons un environnement hiérarchique dans lequel les relations entre les entités sont fortement contraintes en termes d'autorisations et d'authentifications. L'environnement d'une application de paiement en ligne est un exemple d'une situation où la communauté est structurée (il y a des clients et des marchands, les rôles sont bien définis). Par contre, un environnement pair-à-pair peut être considéré comme une communauté d'utilisateurs non structurée. Les relations entre les entités y sont plus souples. Les réseaux sociaux sont de bons exemples de ce type d'environnement.

Notre système est conçu pour pouvoir s'adapter à des applications de domaines variés et en conséquence, il devra satisfaire l'ensemble des critères que nous présentons ci-dessous.

Décentralisation du système : le système devra être décentralisé. Cela concerne plusieurs aspects différents : les preuves, les politiques et les prises de décision sur la

confiance. Si nous imaginons que le système a des millions d'utilisateurs et si de plus ce système est centralisé, il peut devoir traiter simultanément un nombre très important de requêtes, et cela peut poser des problèmes de surcharge. Dans notre système, la politique sera propre à chaque entité et il ne devra pas être nécessaire de disposer de l'ensemble des politiques pour raisonner à leur propos. Si l'on fait référence aux systèmes existants que nous avons étudiés, nous pouvons voir que la plupart d'entre eux sont décentralisés : Trustbuilder [84, 26], SECURE [15, 14], Trust-X [5, 6] et le framework de Krukow [59]. Par contre nous avons pu constater que Keynote [10] et SULTAN[36, 35] utilisent un modèle centralisé. De notre avis, cette décentralisation est nécessaire dans le cas d'une utilisation dans un environnement non structuré où chaque entité peut avoir des profils différents.

Source de la confiance : le système doit prendre en compte toutes les sources d'informations disponibles pour évaluer et établir la confiance. Ces sources d'informations correspondent à une large diversité de types de données : les *credentials* (certificats signés), les informations concernant les risques, l'expérience de l'entité, l'historique de ses transactions, les recommandations...

Nous soulignons que l'utilisation de toutes ces sources d'informations est vue en terme d'agrégations. Par exemple dans un contexte donné, il sera judicieux d'utiliser à la fois des qualifications et des informations concernant les risques, alors que dans un autre contexte, seules les informations sur la réputation seraient pertinentes. Ceci nous offre une grande flexibilité d'adaptation aux applications cibles.

Nous nous fixons cette contrainte de manière à disposer d'un point de vue complet sur l'entité, et ainsi pouvoir évaluer plus précisément son comportement. Nous allons montrer au travers d'un exemple en quoi se restreindre dans les sources d'informations peut rendre moins pertinente l'évaluation de la confiance. Si nous considérons une transaction de e-commerce entre un client et le marchand dans un site en ligne, son montant, s'il est important, va influencer les décisions des partenaires.

- imaginons que le marchand n'utilise que la carte bancaire (comme un *credential*) pour évaluer la confiance en son client et décider d'honorer sa commande. Si cette carte bancaire est utilisée par un *pirate* pour passer une commande dont le montant est important, dans le cadre actuel de la législation française sur la vente à distance, le marchand prend un risque important : celui de ne pas être payé si le propriétaire légitime de la carte dénonce l'achat[1] Dans cette situation, le marchand pourrait utiliser des informations complémentaires, comme son expérience avec ce client ou bien le risque lié à la transaction.
- si le client passe une commande d'un montant faible et qu'il ne veut payer qu'à la réception du produit, le marchand pourrait ne pas avoir besoin de vérifier la validité de la carte bancaire (ce qui peut avoir un coût) mais pourrait seulement se satisfaire des informations de l'historique des commandes de ce client, et de valider sa commande, considérant que cette situation est peu risquée.

Cet exemple montre bien que si nous utilisons de manière flexible toutes les sources d'informations disponibles, le système sera plus efficace dans sa prise de décisions. C'est pourquoi nous devons insister sur ce point dans la conception de notre système de gestion de la confiance.

[1] La législation française protège efficacement les acheteurs dans le cadre de la vente à distance (sur internet par exemple). L'acheteur peut contester tout débit fait dans ce cadre et demander à sa banque de recréditer immédiatement son compte. C'est alors au vendeur de faire la preuve que l'achat a bien été fait par le porteur de la carte bancaire, c'est lui qui supporte les conséquences d'une fraude : il a pu livrer un bien qui ne lui sera jamais payé.

Expression de la politique : le système se compose de plusieurs entités et nous voulons que chacune d'elles puisse exprimer sa propre politique. Notre objectif est de permettre à chaque entité de gérer au mieux ses ressources. Cela correspond également mieux aux modèles des applications actuelles. Pour atteindre cet objectif, le système doit disposer des composants suivants :

- un langage d'expression des politiques bien défini au niveau syntaxique et sémantique. Cela permet aux entités de spécifier facilement leurs politiques et leurs actions.
- le système doit disposer d'un module de vérification pour évaluer la conformité de la politique et des actions, pour répondre aux questions concernant la confiance à associer aux actions entreprises avec des partenaires. La vérification pourra être réalisée par calcul ou par raisonnement.

L'approche que nous avons adoptée pour notre infrastructure de confiance est une approche hybride. Elle dispose des propriétés des approches basées sur les politiques et de celles basées sur la réputation. L'idée principale de notre modèle de confiance est que le langage d'expression des politiques et les mécanismes de raisonnement sur ces politiques pourront prendre en compte toutes les sources d'informations jugées utiles, que ce soient des certificats, les niveaux de risques, la réputation, les recommandations...

Dans notre approche, le niveau de confiance entre les participants est réévalué à chaque étape de la transaction, autrement dit, la confiance est construite de manière itérative et mesurée automatiquement.

Pour ce faire, nous nous basons sur le système de réputation de Krukow *et al.* [59]. Ce système de réputation a été conçu pour évaluer la réputation des entités dans un environnement pair-à-pair. Nous l'adaptons pour notre système de manière à ce qu'il satisfasse les contraintes que nous venons d'énoncer. Un aspect très important de notre système est le mécanisme de négociation de la confiance. C'est ce mécanisme qui permet aux entités du système de négocier pour établir une confiance mutuelle.

La section suivante présente l'architecture générale de notre système de confiance. Les détails sur la formalisation de notre système sont présentés au chapitre 5 et le mécanisme de la négociation de la confiance au chapitre 6.

4.2 Modèle global de gestion de la confiance

Nous proposons notre système de gestion de la confiance dans le cadre d'un contexte distribué où plusieurs entités participent au système. Chaque entité a ses propres ressources et sa propre politique de protection de ces ressources. La question cruciale qui se pose dans le système est : *comment une entité peut-elle faire confiance à une autre entité pour entreprendre une action avec elle ?* Pour cela, nous avons besoin d'un modèle de confiance qui nous permet d'analyser les transactions entre les deux entités à partir des données existantes : la nature de l'action entreprise, les données fournies par l'entité demandant l'action, les informations concernant l'expérience de l'entité gérant les ressources et sa politique.

Le modèle à la base de notre système est inspiré de celui proposé par le projet SECURE [15, 14] que nous avons présenté au chapitre 3. Les auteurs ont utilisé la notion de *type abstrait* [82] pour définir leur modèle théorique des relations de confiance entre les différentes entités de leur système. Avec ces *types abstraits* nous pouvons présenter rigoureusement et formellement le système de confiance : les entités, les relations entre les entités, la politique utilisant un modèle de contrôle d'accès... Nous détaillons ce modèle formel dans la section 3.

L'objectif de notre travail est de disposer d'un modèle opérationnel de gestion de la confiance et de l'utiliser au sein d'applications Internet. Dans le cadre de cette thèse, notre modèle de gestion de la confiance sera illustré par un scénario d'application simple : une transaction de type commerce électronique sur un site de vente en ligne.

L'aspect le plus important est que l'on doit prendre en compte toutes les sources d'informations disponibles pour modéliser les relations entre les acteurs. Nous utilisons les travaux de Krukow *et al.* [59] comme une base de notre système : toutes les informations relatives aux relations entre deux entités sont observées comme des événements ; les événements sont décrits avec le modèle de *structure d'événements* qu'ils ont proposé. Le modèle de confiance est conçu pour être mis en œuvre par chaque entité du système. Nous ajoutons à ce modèle un mécanisme de négociation de la confiance qui permet aux entités qui tentent d'entrer en relation l'une avec l'autre, d'établir étape par étape une relation de confiance mutuelle.

4.2.1 Composants fonctionnels

Le fonctionnement de notre modèle est basé sur la notion de transactions entre deux entités du système. À chaque étape d'une transaction, le système utilise les informations qu'il peut obtenir pour décider s'il poursuit ou non la transaction. Pour le réaliser, nous avons besoin des composants fonctionnels suivants :

Historique des transactions : tous les événements observés par l'entité sur le comportement et les actions de son correspondant, ainsi que sur les recommandations des autres entités, sont conservés dans l'historique des transactions. Cet historique est une séquence de sessions où chaque session est un ensemble chronologique d'événements. Les événements sont le résultat des observations et les sessions représentent donc les transactions à un instant donné de leur déroulement. Pour pouvoir assurer le contrôle d'une transaction, les événements appartenant à une session sont soumis à des contraintes particulières : ne pas entrer en conflit et respecter des liens de causalité. Ces relations sont définies dans une structure de données particulière : la structure d'événements. Cette structure est propre à chaque entité.

Politique de confiance : la politique de confiance spécifie les conditions d'accès aux ressources ou aux services. Le demandeur doit adopter un comportement satisfaisant la politique du fournisseur pour accéder à des ressources. Le fournisseur doit observer un comportement du demandeur qui satisfait sa politique pour fournir la ressource. La politique manipule l'historique des transactions. Le langage de politique que nous utilisons est une logique temporelle (PP-LTL). La politique se présente donc sous la forme d'une formule logique.

Décision sur la confiance : supposons que l'entité demandeur entreprenne une action particulière. Chaque événement observé par l'entité est sauvegardé dans une session de l'historique liée à cette transaction. La vérification de la satisfaction de la politique utilise l'ensemble des informations de l'historique pour déterminer si l'action peut être poursuivie – ce qui nous indique que la transaction se déroule toujours dans des conditions de confiance acceptables.

4.2.2 Architecture

L'architecture générale du système est présentée dans la figure 4.1. Nous en décrivons ci-dessous les composants principaux.

Application Interface : ce composant réalise les observations sur le comportement du demandeur (actions demandées, informations fournies pour la transaction) et des

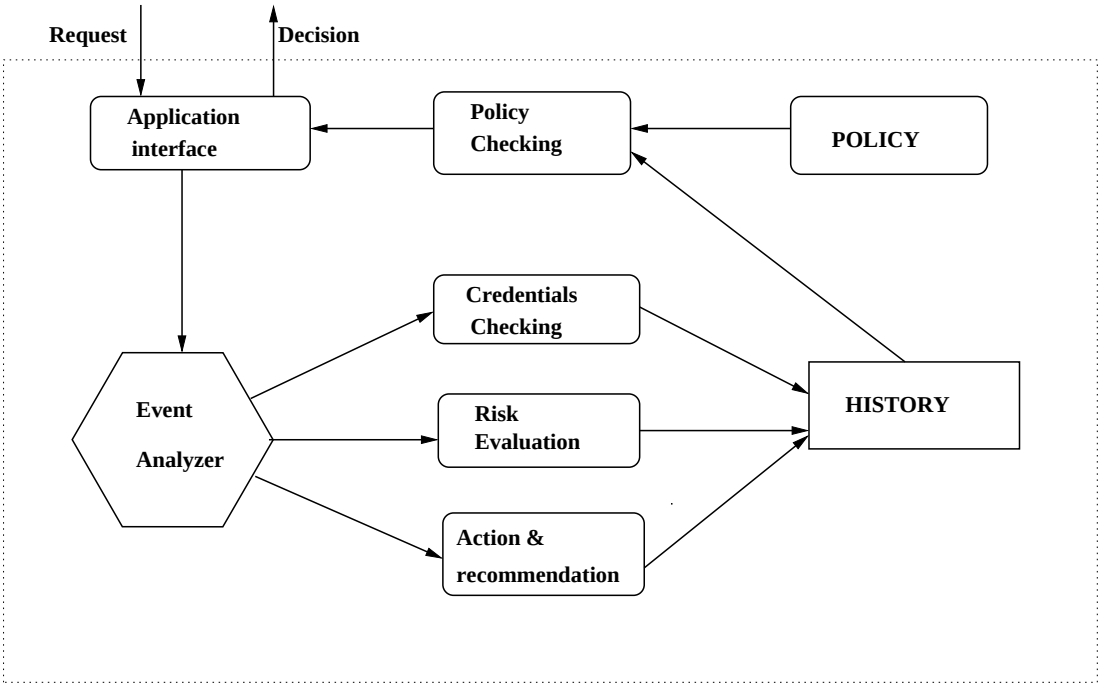


FIG. 4.1 – Système de gestion de la confiance

autres entités concernées dans le système (*i.e. recommandation* pour le demandeur). Il fournit aussi au demandeur l'information de décision de la confiance.

Event Analyzer : ce composant analyse et interprète les observations sur le comportement des entités et fournit à l'entité les événements correspondants. Il n'y a pas de limitation particulière sur le type des événements traités, nous avons cité les *credentials, risques, recommandations, actions...*, qui seront analysés respectivement par les modules *Credentials Checking, Risk Evaluator, Action & Recommendation*. Si cela s'avère nécessaire, l'*Event Analyzer* peut être adapté à des domaines particuliers et interpréter de manière spécialisée les observations qu'il effectue. Il est seulement nécessaire de prendre en compte, lors de la rédaction de la politique, les nouveaux types d'événements.

History : ce composant sauvegarde l'historique des transactions passées de l'entité. Il y a un historique associé à chacune des entités partenaires.

Policy : il s'agit ici de la politique de confiance de l'entité.

Policy Checking : ce module permet de vérifier qu'à chaque instant, la politique de l'entité est satisfiable. C'est ce composant qui va permettre de prendre la décision de faire confiance ou non en une entité pour une action.

Nous allons préciser quelques aspects importants de la conception de notre système par rapport aux systèmes existants :

- Notre système est un modèle *hybride*, il possède à la fois les propriétés des systèmes basés sur les politiques et de ceux basés sur l'expérience. Il utilise un historique qui lui permet de conserver des informations relatives à l'expérience, la recommandation... pour prendre ses décision. Les décisions sur la confiance sont conçues après une vérification de la politique en utilisant toutes les sources d'informations concernées.
- Les travaux de Krukow [59] tels qu'ils sont présentés ne s'appliquent qu'aux systèmes de réputation. Notre système peut être considéré comme une extension de

ce travail, étendu par un mécanisme de négociation de la confiance entre deux acteurs du système. Ce résultat est donné dans le chapitre 5.
– Dans le système de gestion de la confiance SULTAN[35, 36], ce point « hybride » est également considéré : il prend en compte toutes les informations disponibles telles que le risque, la recommandation et les credentials pour évaluer la confiance. Mais le problème est qu'il est centralisé : toutes les sources de données telles que les preuves, les qualifications et les politiques sont stockées dans une base centrale. Le raisonnement sur la confiance est également réalisé de manière centralisée.

4.3 Modèle de confiance de SECURE

Dans cette section, nous présentons le modèle formel de la confiance, proposé dans le cadre du projet SECURE [15, 14]. Ce modèle décrit de manière théorique comment la confiance peut être acquise par les entités pour prendre les décisions sur les actions concernées. Notre système se présente comme une instance (un cas concret) du modèle formel de SECURE.

Le modèle de confiance de SECURE est à considérer dans un environnement global qui se compose de plusieurs *principals*. Chaque *principal* dispose d'une valeur de confiance dans les autres *principals*. Cette valeur peut être mise à jour lorsque de nouvelles informations sont disponibles. Une décision concernant la confiance d'un *principal* en un autre *principal* peut utiliser les informations fournies par les autres *principals*. La valeur de la confiance entre les *principals* du système peut être déterminée soit par calculs, soit par raisonnements.

Soit P l'ensemble de tous les *principals* du système et soit T l'ensemble des valeurs de confiance entre ces *principals*. Le *framework* de confiance SECURE est défini en supposant que nous disposons d'une *boîte noire* de confiance. Cette boîte noire est un outil intégré au système et elle permet de déterminer et de mettre à jour les relations de confiance entre les *principals*.

À chaque instant, une boîte noire se trouve dans un état particulier, état appartenant à l'ensemble des états S. Un ensemble d'événements (interactions du *principal*) E permet de mettre à jour les états de la boîte noire.

Le formalisme utilisé par les auteurs de SECURE est repris des travaux de formalisation des systèmes de confiance de S. Weeks [82]. P est décrit comme étant le type abstrait des acteurs et T le type du niveau de confiance qu'ils entretiennent. Le type $P \rightarrow P \rightarrow T$ est défini comme étant celui d'une fonction calculant la confiance entre deux acteurs.

Les relations de confiance entres les entités sont définies de la manière suivante :
– La confiance est une fonction m qui a comme paramètres deux acteurs de P et qui fournit comme résultat une valeur de confiance appartenant à T. Cette relation peut s'écrire :

$$m : P \rightarrow P \rightarrow T$$

La fonction m appliquée à a et b rend la valeur de confiance $m(a)(b) \in E$ qui exprime la confiance que a a en b ; a et b sont des acteurs du système.
– Raisonnements : la modification de l'état de la boîte noire est défini par l'assertion *update* du type $S \times E \rightarrow S$ (l'observation d'un événement modifie l'état de la boîte noire) et la confiance dans un *principal* en fonction de l'état de la boîte noire est définie par l'assertion *trust* de type $S \times P \rightarrow T$.

– La politique d'un acteur a est définie par la confiance qu'il accorde aux autres acteurs :

$$\pi_a : [P \to P \to T] \to [P \to T]$$

Il s'agit d'une politique locale au *principal* a. Cette politique exprime le fait que la confiance que l'on accorde à un *principal* peut être conditionnée par la confiance que s'accordent mutuellement d'autres entités du système (je fais confiance à c si je fais confiance à b et que b fait confiance à c par exemple).

– La politique de confiance globale du système est définie par :

$$\Pi : [P \to P \to T] \to [P \to P \to T]$$

Ce modèle formel d'un système de gestion de la confiance nous en donne seulement une vision théorique. Les entités, les relations et les politiques ont besoin de prendre une forme opérationnelle pour pouvoir être mises en œuvre dans des applications pratiques. Notre système de gestion de la confiance, qui est basé sur ce modèle, est développé dans ce sens.

4.4 Logique temporelle linéaire - LTL

Nous présentons dans cette section une logique temporelle linéaire (LTL). Cette présentation nous permet d'introduire une variante de cette logique, PP-LTL (Pure Past Linear Temporal Logic) qui est dédiée aux faits passés. Une présentation plus détaillée peut être trouvée dans [57].

Une logique temporelle est une extension de la logique conventionnelle : elle intègre de nouveaux opérateurs qui expriment la notion du temps. La logique classique ne peut pas exprimer une assertion liée au comportement d'un système par exemple, « après exécution d'une instruction I, le système se bloque ». Dans une telle assertion, les actions s'exécutent en suivant l'axe du temps : à l'instant t l'instruction I est exécutée et à l'instant $t+1$ le système est bloqué. Les logiques temporelles ont été proposées pour modéliser ce type d'assertion, en fournissant des opérateurs pour exprimer les comportements passés et futurs du système.

La logique LTL permet de représenter le comportement des systèmes réactifs au moyen de propriétés. Les propriétés spécifient le comportement du système à un moment de son exécution.

Le système fonctionne de manière linéaire dans le temps. Les exécutions du système peuvent être modélisées par un système de transition ST qui se compose de plusieurs états, et permet de passer de l'un à l'autre. Chaque état de ST correspond à un état du système à un instant t. À chaque état, nous pouvons observer les propriétés, si elles existent, pour identifier le comportement du système. Le comportement global du système est spécifié par l'observation des propriétés dans tous les états concernés.

L'activité du système se déroule de manière chronologique (figure 4.2). Elle se décompose en une séquence d'états représentant les actions de l'entité dans le temps.

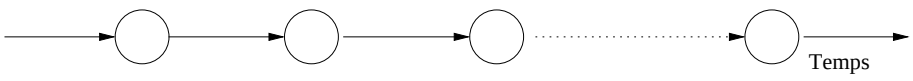


FIG. 4.2 – Déroulement temporel d'un système en logique LTL

4.4.1 Syntaxe

Les formules de la logique LTL peuvent être définies de la manière suivante :
- Propositions atomiques :
  $true \mid false \mid (x = v)$ sont des formules ($(x = v)$ se lit x a pour valeur v).
- Connecteurs booléens :
  Soit $\phi, \psi \in LTL$, toutes les formules de la forme suivante appartiennent à LTL :
  $\neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \phi \leftrightarrow \psi$
- Connecteurs temporels :
  Soit $\phi, \psi \in LTL$, toutes les formules de la forme suivante appartiennent à LTL :
  $G\phi \mid F\phi \mid \phi\, U\, \psi \mid X\phi$

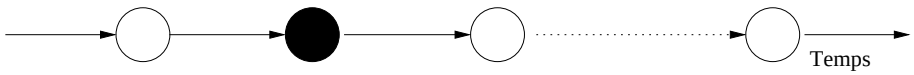
4.4.2 Sémantique

Soit $\Pi = s_0, s_1, s_2 \ldots$, une séquence d'états du système de transition, et soit ϕ une formule en logique LTL. On définit « ϕ est satisfaite pour Π », noté $\Pi \models \phi$ de la manière suivante : pour tout $i = 0, 1 \ldots$, on désigne par Π_i la séquence d'états $s_i, s_{i+1}, s_{i+2} \ldots$ (on note que $\Pi = \Pi_0$), soit ϕ et ψ des formules en logique LTL.

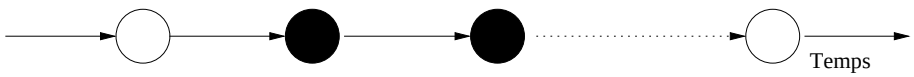
1. $\Pi \models true$ et $\Pi \not\models false$
2. $\Pi \models (x = v)$ ssi $s_0 \models (x = v)$.
3. $\Pi \models \phi \wedge \psi$ ssi $\Pi \models \phi$ et $\Pi \models \psi$
   $\Pi \models \phi \vee \psi$ ssi $\Pi \models \phi$ ou $\Pi \models \psi$
4. $\Pi \models \neg\phi$ ssi $\Pi \not\models \phi$
5. $\Pi \models \phi \rightarrow \psi$ ssi $\Pi \not\models \phi$ ou $\Pi \models \psi$
   $\Pi \models \phi \leftrightarrow \psi$ ssi $\Pi \not\models \phi$ et $\Pi \not\models \psi$ à la fois, ou $\Pi \models \phi$ et $\Pi \models \psi$ à la fois.
6. $\Pi \models X\phi$ si $\Pi_1 \models \phi$
   $\Pi \models F\phi$ si $\exists i \in (0, 1, \ldots)$ tel que $\Pi_i \models \phi$
   $\Pi \models G\phi$ si $\forall i \in (0, 1, \ldots)$ tel que $\Pi_i \models \phi$
   $\Pi \models \phi U \psi$ si $\exists k \in (0, 1, \ldots)$ tel que $j < k, \Pi_j \models \phi$ et $j \geq k, \Pi_j \models \psi$

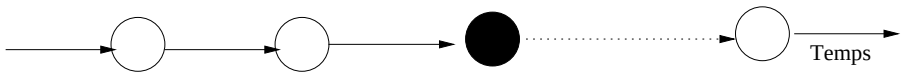
Nous explicitons la sémantique des opérateurs temporels $\boldsymbol{X}$, $\boldsymbol{F}$, $\boldsymbol{G}$ et $\boldsymbol{U}$ avec les figures suivantes (les états en noir indiquent que la formule est vérifiée) :

1. $\boldsymbol{X}$: $X\ \phi$ indique que ϕ est vérifiée à l'état suivant.

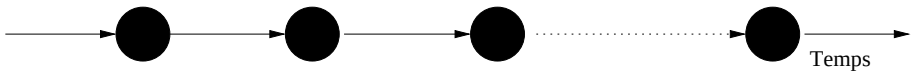


2. $\boldsymbol{F}$: $F\ \phi$ indique qu'il existe un état pour lequel ϕ sera vérifiée.

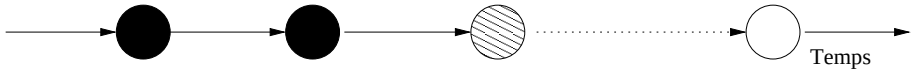




3. $\boldsymbol{G}$: $G\ \phi$ indique que ϕ est toujours vérifié dans le futur.



4. ***U*** : $\phi U \psi$ indique que ϕ est vérifié jusqu'à ce que ψ le soit.



Notons que l'état hachuré indique à partir de quel moment ψ est vérifiée.

Nous pouvons remarquer que la logique LTL est utilisée pour spécifier des faits (le comportement d'une entité) dans le temps (passé et futur). Pourtant, de par les particularités de notre modèle de gestion de la confiance, nous n'avons besoin que d'exprimer des comportements passés. Nous utiliserons donc dans la suite de nos travaux une logique temporelle qui ne s'intéresse qu'aux faits passés : PP-LTL (Pure-Past LTL). Nous présenterons à la section 4.5.3 les détails de PP-LTL.

4.4.3 Exemple

Supposons un système de transactions qui modélise le système téléphonique. Nous avons les deux événements *signal* et *réponse* qui représentent la signalisation d'un appel (sonnerie) et l'actions d'y répondre. Pour exprimer le fait que *tout signal est suivi d'une réponse*, nous pouvons utiliser la formule suivante :

$$G(signal \rightarrow F\ reponse)$$

4.4.4 Vérification

Les techniques du *model checking* [17] sont traditionnellement utilisées pour la vérification des formules de logique LTL. Les algorithmes de vérification connus sont *NP-complet*.

4.5 Framework logique pour les systèmes de réputation

Dans cette section, nous détaillons le fonctionnement du framework logique défini pour le mécanisme de réputation proposé par Karl Krukow *et al.* [59]. Ce système de réputation a été conçu pour les applications en ligne qui ont besoin de la confiance pour contrôler les interactions entre les différentes entités du système. K. Krukow a proposé un modèle d'application qui repose sur une *structure d'événement* qui formalise le système par son utilisation des événements et des politiques pour chaque entité.

Le contenu de cette section est un résumé du travail de K. Krukow *et al.* [59] auquel nous avons ajouté les détails qui permettront au lecteur de mieux comprendre de quelle manière nous l'avons adapté à notre système de négociation de la confiance. Pour finir ce préambule, nous rappelons que le mécanisme de recommandation proposé par K. Krukow n'est en aucun cas un mécanisme de négociation.

Le principe qui à été mis en œuvre lors de la conception du framework de recommandation de K. Krukow est le suivant : *utiliser les informations sur les comportements passés du* principal *pour évaluer de futures interactions avec lui.*

Dans ce modèle, les informations sur les comportements passés et futurs des participants sont spécifiées dans le langage de politique. Tous les comportements passés d'un *principal* sont conservés dans un historique des transactions.

Toute action future entreprise par un *principal* sera validée si nous avons la conformité de la politique et de l'historique des transactions. Le contrôle d'accès aux ressources est réalisé de la manière suivante : « si le *principal* p a le droit d'accéder à la ressource r

à l'instant t, alors les comportements passés de p jusqu'à l'instant t satisfont la politique d'accès à r ».

Le *principal* observe les comportements passés de son partenaire sous la forme d'événements. L'historique qui contient les informations sur ces comportements est donc un ensemble d'événements respectant les règles de relation entre événements. La modélisation de ces relations entre événements est réalisée a l'aide de la *structure d'événements* (event structure) [71].

Le langage proposé pour exprimer la politique d'interaction des entités est une variante de la logique LTL [74], appelée PP-LTL (Pure-Past LTL). Cette logique est restreinte à l'expression des faits survenus dans le passé par l'usage des quantificateurs G^{-1}, F^{-1} et X^{-1} dont la sémantique est définie en logique LTL. Dans ce modèle, le comportement est modélisé comme un ensemble d'événements et chaque structure linéaire peut être modélisée par une structure d'événements que nous allons présenter ci-dessous.

Chaque entité dans le système a sa propre politique pour protéger ses ressources. Nous pouvons proposer la politique de l'entité en utilisant un modèle d'autorisation d'accès à la ressource. Ce modèle d'autorisation d'accès nous permet de définir les conditions, les contraintes pour avoir accès à la ressource. C'est une base pour définir la politique de l'entité.

De manière à contrôler l'accès aux ressources, la politique de sécurité doit être évaluée en fonction de l'historique des comportements de l'entité. Cela se ramène à un problème de vérification de la satisfaction d'une formule logique PP-LTL auprès de l'historique de l'entité. Les auteurs ont utilisés les algorithmes proposés par Havelund et Roçu [40] pour résoudre ce problème de vérification. Il s'agit d'un algorithme basé sur les techniques de la programmation dynamique et qui, bien qu'efficace n'en reste pas moins de complexité exponentielle (il s'agit d'un problème *NP-complet*).

4.5.1 Observation des événements

Le système est composé de plusieurs entités en interaction. Ces entités s'échangent des informations sous la forme de messages. L'observation de ces messages permet aux entités de déterminer les événements qui seront à prendre en compte et à rajouter à l'historique. Si une entité a des interactions avec plusieurs autres entités au même moment, les observations doivent permettre de répartir les événement dans les historiques concernés (notion de sessions indépendantes).

Voici un exemple de ce fonctionnement appliqué à un scénario d'enchères sur le site eBay [38].

Lorsqu'un vendeur décide de vendre un produit aux enchères, il commence par publier une annonce sur le site. Á partir de ce moment, les clients potentiellement intéressés par ce produit peuvent commencer à enchérir et à sur-enchérir. Lorsque la période d'enchère est terminée (en général une ou deux semaines selon les préférences du vendeur), le client qui a fait l'offre la plus élevée remporte l'enchère. Ce client va maintenant avoir à effectuer la transaction avec le vendeur du produit pour concrétiser son achat. Le module de gestion de la confiance est réalisé de manière à lui permettre d'observer les événements et de prendre des décisions pour chacune des étapes de la transaction.

Maintenant que l'enchère est terminée, nous spécifions *la transaction* par l'ordre des différents échanges entre le client et le vendeur sous la forme du protocole suivant :

– Le vendeur envoie *la confirmation* de la vente du produit objet de l'enchère et en *demande de paiement.*

– Lorsque le client reçoit la *demande de paiement*, il a deux choix possibles : payer pour continuer la transaction et disposer du produit ou ne pas payer. Si le client ne paie pas le vendeur, la transaction se termine. Si le client paie, la transaction se poursuit.
– Lorsque le vendeur reçoit le paiement, il envoie une confirmation de *paiement* au client et lui envoie le produit. À ce stade de la transaction, le vendeur peut être malhonnête et ne pas envoyer la confirmation de paiement ni le produit au client.
– Enfin, le vendeur et le client peuvent évaluer mutuellement la transaction (*positive, neutral, negative*).

Le client peut observer les événements suivants concernant le comportement du vendeur : *la demande de paiement*, *le paiement au vendeur*, *la confirmation du paiement* par le vendeur (et bien sûr l'envoi du produit) ou bien le fait d'ignorer la demande de paiement et enfin, *l'évaluation de la transaction.*

Le scénario sera modélisé dans un framework de la structure d'événements expliqué en détail dans les sections suivantes. Cela donnera différents éléments du scénario : des événements observés possibles, l'historique de la transaction et la politique de sécurité appliquée.

4.5.2 Framework de structure des événements

Quand une interaction a lieu, nous voyons apparaître les événements dans un certain ordre et il existe des relations entre eux qui sont liées au type de transaction qui est en train de se dérouler. Certains événements ne peuvent pas se produire si d'autres se sont déjà produit. Nous dirons que ces événements sont en *conflit* et qu'il y aura une exclusion mutuelle entre ces événements. Par exemple, dans le scénario de l'application *eBay*, si le *client* a généré l'événement *positive* (l'évaluation du vendeur sur la transaction), on ne peut pas observer l'événement *neutral* ou l'événement *negative* sur cette transaction. Un événement peut *dépendre* d'un autre dans le sens où ce deuxième événement ne pourrait pas avoir lieu si le premier n'avait pas été déjà observé (relation de causalité). Dans le scénario *eBay*, l'événement de *confirmation* de paiement ne peut pas avoir lieu si l'événement *paiement* n'a pas eu lieu au préalable. Les événements sont dits *indépendants* s'ils ne sont ni en *conflit* ni *dépendant* d'autres événements. Par exemple les événements *paiement* et *négative* (évaluation de la transaction) sont indépendants.

Un événement ne peut être rajouté à l'historique d'une transaction que s'il n'entre pas en conflit avec un événement déjà observé et qu'il respecte les conditions de dépendances imposées par l'application. La notion de *structure d'événements* va nous permettre de spécifier ces contraintes.

Structure d'événements

Une structure d'événement est un triplet $ES = (E, \leq, \#)$ qui se compose d'un ensemble d'événements E et de deux relations binaires sur E : $\leq$ et $\#$. Chaque élément $e \in E$ est un événement. La relation $\#$ est la relation de conflit, et $\leq$ est la relation de causalité.

– Relation de causalité ($\leq$) : étant donné un événement $e' \in E$, pour tout $e \in E$, si $e \leq e'$ (e' dépend/est une conséquence de e) alors e' ne peut apparaître qu'après e ; pour chaque $e \in E$, l'ensemble $\lceil e \rceil = \{ e' \in E \mid e' \leq e \}$ est fini. Cette relation est symétrique et non-réflexive.
– Relation de conflit ($\#$) : cette relation doit satisfaire l'axiome de transitivité suivant : pour tout événement $e, e', e" \in E$, ($e \# e'$ *et* $e' \leq e"$) implique $e \# e"$.

– Événements indépendants : deux événements sont indépendants s'ils n'ont pas de relation de causalité et ne sont pas en conflit.

La figure 4.3 présente une structure d'événements utilisable dans le cadre du scénario de l'application eBay que nous venons de décrire dans le paragraphe précédent.

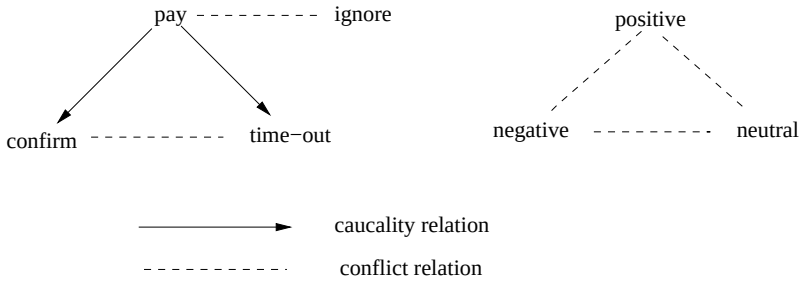


FIG. 4.3 – Structure d'événements observée par *l'acheteur*

Les evénements observés par l'acheteur représentent les différents comportements que peut avoir l'entité. Ils ont le sens suivant :

– *positive, negative, neutral* : il s'agit des valeurs possibles que peut prendre l'évaluation d'une transaction. Lorsque la transaction est terminée (par un échec ou pas), l'acheteur peut donner une évaluation sur la transaction.
– *pay/ignore* : l'événement *pay* indique que l'acheteur paie pour la transaction après avoir reçu la demande de paiement du vendeur. Par contre, *ignore* indique que l'acheteur ignore, la transaction sera interrompue.
– *confirm/time-out* : les événements indiquant le comportement du vendeur après que l'acheteur a payé pour sa demande de paiement. Soit le vendeur confirme avoir reçu le paiement (*confirm*), soit il ne le confirme pas (*time-out*). Dans ce cas de *time-out*, le client considère que le vendeur est malhonnête et la transaction échoue.

Configuration

Avec les relations de *causalité* et de *conflit*, nous pouvons construire un sous-ensemble des événements observables. Les événements observés respectant ces relations représentent une session de la transaction et sont désignés sous le nom de *configuration* :

Soit une structure d'événements $ES = (E, \leq, \#)$. Un sous-ensemble d'événements $X \in E$ est une *configuration*, si et seulement si, il satisfait les conditions d'absence de *conflit* et *de causalité* fermée (clôture de E par $\leq$). Les propriétés suivantes doivent être vérifiées : Pour tout $d, d' \in X$ et $e \in E$

– absence de conflit : $d \neg \# d'$
– causalité fermée : $e \leq d \Rightarrow e \in X$

La notation C_{ES} désigne l'ensemble des configurations de ES, et $C^0_{ES} \subseteq C_{ES}$ signifie un ensemble fini des configurations. Une *configuration* est maximale si son ordre partiel $(C_{ES}, \subseteq)$ est maximal. Une configuration finie permet de modéliser une transaction. La configuration maximale présente l'information complète d'une transaction. L'entité peut avoir plusieurs transactions et elle doit observer et prendre en compte l'information de toutes ces transactions.

Historique

Pour une structure d'événements ES, l'historique local des transactions est un ensemble fini de configurations, $h = x_1 x_2 \dots x_n \subseteq {C^0_{ES}}^*$. Chaque élément x_i dans l'historique h est une session qui correspond à l'exécution d'une transaction.

Dans le cas du scénario *eBay* et en utilisant la structure d'événements présentée figure 4.3, nous pourrions avoir l'historique suivant :
`{pay,confirm,positive}{pay,confirm,neutral}{pay}`
Cet exemple nous montre l'historique d'un client qui a eu trois transactions avec un même vendeur. Les deux premières transactions sont terminées, la troisième en est à l'étape du *paiement* et en attente de sa confirmation.

Les deux opérations suivantes sont définies pour manipuler l'historique : **new** et **update**.

- L'opération **new** : ${C^0_{ES}}^* \rightarrow {C^0_{ES}}^*$, **new**(h)=$\emptyset$.
- L'opération **update** : ${C^0_{ES}}^* \times E \times N \rightarrow {C^0_{ES}}^*$
  Pour tout $h = x_1 x_2 \dots x_n \subseteq {C^0_{ES}}^*, e \in E, i \in N$, **update**$(h, e, i)$ est définie si $i \in \{1, 2, \dots, n\}$ et $x_i \rightarrow x_i \cup \{e\}$: **update**$(h, e, i) = x_1 x_2 \dots (x_i \cup \{e\}) \dots x_n$.

4.5.3 Langage de politique

Le fait de disposer d'informations sur le comportement passé de l'acteur nous permettra d'en tenir compte lorsqu'il sera nécessaire de prendre des décisions lors d'interactions futures. Les décisions qui seront à prendre dans ce contexte sont de type binaire, elles indiqueront si *l'interaction* pourra avoir lieu ou non. la décision de continuer la transaction sera prise si la politique locale peut être satisfaite en utilisant l'historique des interactions.

Dans ce système, l'auteur a utilisé un langage d'expression de la politique qui est une variante de LTL [74], variante qui ne s'applique qu'aux seuls faits passés, logique PP-LTL. Dans ce langage, l'historique des transactions est basé sur la structure d'événements et est utilisé comme un système de transitions pour tirer le meilleur parti des algorithmes issus du *model-checking*.

La politique est représentée par une formule de logique PP-LTL. On définit la relation de satisfaction $\models$ entre une politique ψ et un historique H : $H \models \psi$ (l'historique H satisfait aux exigences de la politique ψ).

La politique utilisée dans ce système est construite de manière à assurer le contrôle d'accès aux ressources. Pour construire leur modèle, les auteurs se sont inspirés du modèle de contrôle d'accès utilisant l'historique des comportements de Edjlali *et al.* [24].

Syntaxe

Le langage est défini pour une structure d'événements $ES = (E, \leq, \#)$. Supposons que e, e', e_i soient des événements appartenant à E. La syntaxe du langage, donnée sous sa forme *BNF* est la suivante :

$$\psi \ ::= \ e \ | \ \diamond e \ | \ \psi_0 \ \vee \ \psi_1 \ | \ \psi_0 \ \wedge \ \psi_1 \ | \ \neg e \ | \ X^{-1}\psi \ | \ \psi_0 \ S \ \psi_1$$

Pour cette définition, e et $\diamond e$ sont des propositions atomiques. Un événement e est vrai dans une session si e y est observé. La proposition $\diamond e$ (e *est possible*) est *vrai* si l'événement e est encore observable dans cette session. Les opérateurs X^{-1} (à l'étape précédente - *last time*) et S (depuis - *since*) sont les opérateurs standards concernant

les actions passées. La formule logique $X^{-1}\psi$ est vraie si ψ est vraie dans la session précédente. La formule logique $\psi_0\ S\ \psi_1$ est vraie si ψ_1 a été vraie dans une session antérieure et si ψ_0 est vraie dans toutes les sessions depuis ce moment là.

Sémantique

La sémantique du langage est présentée ici de façon inductive en utilisant la relation de satisfaction. L'historique contient plusieurs sessions $H = x_1x_2\dots x_N \subseteq {C^0_{ES}}^*$ avec $N > 0$. La sémantique n'est considéré que pour les historiques non-vides ($H \in {C^0_{ES}}^+$). La relation de satisfaction $H \models \psi$ implique que l'historique H satisfait les exigences de la politique ψ pour toutes les sessions de H. Nous noterons $(H, i) \models \psi$ le fait que H restreint à ses i premières sessions satisfait ψ. Nous avons donc $H \models \psi$ si et seulement si $(H, \mid H \mid) \models \psi$ (ssi ψ est vérifiée pour tous les éléments de H, $\mid H \mid$ est le cardinal de H). Nous donnons ici la définition inductive de cette relation de satisfaction :

La vérification de $(H, 1) \models \psi$ est définie la manière suivante :

$$
\begin{array}{lll}
(H,1) \models e & \text{ssi} & e \in x_1 \\
(H,1) \models \diamond e & \text{ssi} & e\neg\#x_1 \\
(H,1) \models \psi_0\ \wedge\ \psi_1 & \text{ssi} & (H,1) \models \psi_0 \text{ and } (H,1) \models \psi_1 \\
(H,1) \models \psi_0\ \vee\ \psi_1 & \text{ssi} & (H,1) \models \psi_0 \text{ ou } (H,1) \models \psi_1 \\
(H,1) \models \neg\psi & \text{ssi} & (H,1) \not\models \psi \\
(H,1) \models X^{-1}\psi & \text{ssi} & false \\
(H,1) \models \psi_0\ S\ \psi_1 & \text{ssi} & (H,1) \models \psi_1
\end{array}
$$

Si $1 < i \leq N$ et si $(H, i-1) \models \psi$ est définie pour ψ, alors on peut définir $(H, i) \models \psi$ de la manière suivante :

$$
\begin{array}{lll}
(H,i) \models e & \text{ssi} & e \in x_i \\
(H,i) \models \diamond e & \text{ssi} & e\neg\#x_i \\
(H,i) \models \psi_0\ \wedge\ \psi_1 & \text{ssi} & (H,i) \models \psi_0 \text{ and } (H,i) \models \psi_1 \\
(H,i) \models \psi_0\ \vee\ \psi_1 & \text{ssi} & (H,i) \models \psi_0 \text{ ou } (H,i) \models \psi_1 \\
(H,i) \models \neg\psi & \text{ssi} & (H,i-1) \not\models \psi \\
(H,i) \models X^{-1}\psi & \text{ssi} & false \\
(H,i) \models \psi_0\ S\ \psi_1 & \text{ssi} & (H,i) \models \psi_1 \text{ ou} \\
 & & (((H,i-1) \models \psi_0\ S\ \psi_1) \text{ and } ((H,i) \models \psi_0))
\end{array}
$$

Ces définitions inductives de la sémantique sont équivalentes à la sémantique courante de la logique LTL. Cette sémantique est définie seulement pour les historiques non-vides, $h \in {C^0_{ES}}^+$. Cela veut dire que la politique ne peut pas être interprétée correctement si nous n'avons jamais eu de transaction. En pratique, l'acteur aura besoin de prendre explicitement une décision s'il se trouve dans cette situation.

Voici quelques formes équivalentes utiles à la rédaction de politiques :

$$
\begin{array}{rcl}
false & \equiv & e\ \wedge\ \neg e \text{ pour tout } e \in E, \\
true & \equiv & \neg false, \\
\psi_0 \rightarrow \psi_1 & \equiv & \neg\psi_0 \vee \psi_1, \\
F^{-1}(\psi) & \equiv & true\ S\ \psi \ (\psi \text{ a été vraie au moins une fois par le passé}), \\
G^{-1} & \equiv & \neg F^{-1}(\neg\psi) \ (\psi \text{ est toujours vraie dans le passé}), \\
\sim e & \equiv & \neg \diamond e \ (e \text{ est impossible}).
\end{array}
$$

Exemples

Nous considérons à nouveau le scénario d'enchères sur eBay qui a été présenté dans la section précédente. En utilisant la structure d'événement présentée figure 4.3, nous pouvons exprimer les politiques suivantes :

- **Politique 1** : le client ne participe qu'à des enchères proposées par les vendeurs qui ont toujours envoyé les produits payés.
  $\psi^{bid} \equiv \neg F^{-1}(time - out)$
- **Politique 2** : le client indique en plus que le vendeur n'a jamais émis d'avis *"négatif"* sur une enchère dont le paiement a été réalisé.
  $\psi^{bid} \equiv \neg F^{-1}(time - out) \wedge G^{-1}(negative \rightarrow ignore)$

4.5.4 Vérification de la satisfaction des politiques

L'observation des interactions permet au système de construire l'historique H. Nous avons besoin de savoir si cet historique respecte la politique ψ. Nous avons donc à vérifier que H satisfait bien la politique ψ ($H \models \psi$, pour $H \in {C^0_{ES}}^+$). D'un point de vue dynamique, lors de chaque décision, il est nécessaire de vérifier que notre politique est toujours satisfaite. Pour cette raison il est nécessaire de disposer d'un mécanisme de vérification efficace.

Dans le cas général, la complexité associée à la vérification et à la satisfiabilité d'une formule de logique LTL est un problème NP-complet [80] (PSPACE-complete pour être précis). Par contre, dans le cas de la logique LTL restreinte aux faits passés, nous allons voir qu'il existe des algorithmes efficaces pour vérifier la satisfiabilité d'une solution partielle.

Krukow *et al.* [59] ont proposé un modèle de vérification dynamique appelé *DMC* (Dynamic Model Checking). Les implémentations proposées utilisent l'algorithme récursif proposés par Havelund et Ro su[40].

Le principe de cet algorithme est le suivant : la vérification de la satisfaction de ψ pour la session (H, m) ($(H, m) \models \psi$) peut se transformer en un processus récursif

1. par la vérification de toutes les sous-formules ψ_j de ψ, pour la session précédente ($(H, m-1) \models \psi_j$),
2. et par la vérification de toutes les sous-formules propres ψ_i de ψ de la session courante, ($(H, m) \models \psi_i$). Une sous-formule de ψ est dite *propre* s'il ne s'agit pas de ψ elle-même.

Les auteurs ont proposé deux implémentations de cet algorithme qui sont basées soit sur l'utilisation d'un tableau, soit sur l'utilisation d'un automate.

La complexité de chacun de ces algorithmes est calculée en fonction de la taille de la politique ($\mid \psi \mid$), de la taille de la structure d'événements utilisée ($\mid C_{ES} \mid$) et du nombre de sessions actives dans l'historique (K).

Implémentation utilisant un tableau

- Initialisation de la structure de données : $O(\mid \psi \mid)$,
- vérification : $O(1)$,
- création d'une session : $O(\mid \psi \mid)$,
- mise à jour d'une session : $O((K - i + 1). \mid \psi \mid)$,
- complexité en espace : $O(K.(\mid \psi \mid + \mid E \mid)$.

Implémentation utilisant un automate à états fini

- Initialisation de la structure de données : $O(2^{|\psi|}. \mid C_{ES} \mid . \mid \psi \mid)$,
- vérification : $O(1)$,
- création d'une session : $O(\mid \psi \mid)$,
- mise à jour d'une session : $O((K - i + 1))$,
- complexité en espace : $O(K. \mid E \mid +2^{|\psi|}. \mid C_{ES} \mid)$.

4.5.5 Discussions

Ce langage d'expression de politique peut être rendu plus flexible ; deux quantificateurs supplémentaires ont été suggérés par l'auteur : $\exists$ et $\forall$. À l'aide de ces deux quantificateurs, les politiques pourront être spécifiées de façon plus simple.

Considérons l'exemple suivant : soit deux événements $open\_f$ et $create\_f$ qui représentent respectivement les actions *ouvrir* et *créer* le fichier *f*. La formule $G^{-1}(open\_f \rightarrow F^{-1}create\_f)$ exprime le fait que le fichier f doit être créé avant de pouvoir être ouvert. Si nous voulons pouvoir utiliser cette politique pour n'importe quel fichier, nous pourrions l'indiquer avec le quantificateur $\forall$ si nous en disposions :

$$G^{-1}(\forall x.[open(x) \rightarrow F^{-1}create(x)])$$

4.6 Synthèse

Dans ce chapitre, nous avons présenté les principes de conception de notre infrastructure de gestion de la confiance. Nous avons d'abord indiqué quels étaient les critères que nous avons retenus dans notre approche : *décentralisation, utilisation de plusieurs sources d'informations, expression de la politique...* Nous avons également présenté l'architecture globale de notre système de gestion de la confiance. Par la suite, nous avons présenté les éléments principaux qui constituent les bases qui nous permettront de construire notre système de confiance : le *modèle formel de confiance de SECURE*, *la logique LTL* et *le modèle de la structure des événements.*

Chapitre 5

Notre proposition pour un système de gestion de la confiance

Dans ce chapitre, nous présentons notre modèle de système de gestion de la confiance dont les grands principes ont été donnés au chapitre 4. Nous nous intéressons d'abord à la formalisation du système avec ses différents composants impliqués dans les procédures de décisons liées à la confiance. Le module de négociation de la confiance sera abordé au chapitre 6.

Pour démontrer l'intérêt de notre modèle, nous présentons deux scénarios d'applications : un système de confiance pour le commerce électronique et un système de confiance pour la gestion de l'évolution des documents collaboratifs [1]. Le système pour le commerce électronique a pour objectif de gérer une transaction et à chaque instant d'indiquer au client ou au vendeur si la transaction peut être poursuivie en toute confiance. Le système de gestion de l'évolution de documents collaboratifs permet d'évaluer si la contribution à un article est pertinente ou non.

5.1 Architecture

Nous rappelons ici l'architecture générale de notre système de confiance, qui a été abordée dans le chapitre 4. Elle est illustrée par la figure 5.1.

Comme nous pouvons le voir, cette architecture dispose de plusieurs composants : *Application Interface, Event Analyzer, History, Policy Checking et Policy*. Nous allons détailler le fonctionnement de chacun de ces éléments. Lorsque l'entité reçoit une sollicitation extérieure, elle est prise en compte par l'*Application Interface*, Les données observées peuvent contenir différents types d'informations telles que des *qualifications*, des recommandations...

A l'étape suivante, ces données sont traitées par le module *Event Analyzer* et ses sous-composants pour être transformées en événements. Les sous-composants (*Credentials Checking, Actions, Risk manager*...) peuvent engendrer des événements complémentaires en fonction des données observées : par exemple une demande de paiement pour un montant élevé peut déclencher la création des événements *paiement_request* et *high_risk*.

Ensuite, tous ces événements sont sauvegardés dans la session concernée de l'*History* et le module *Policy Checking* vérifie que l'historique ainsi mis à jour satisfait toujours

[1] http://fr.wikipedia.org/wiki/Wikipédia

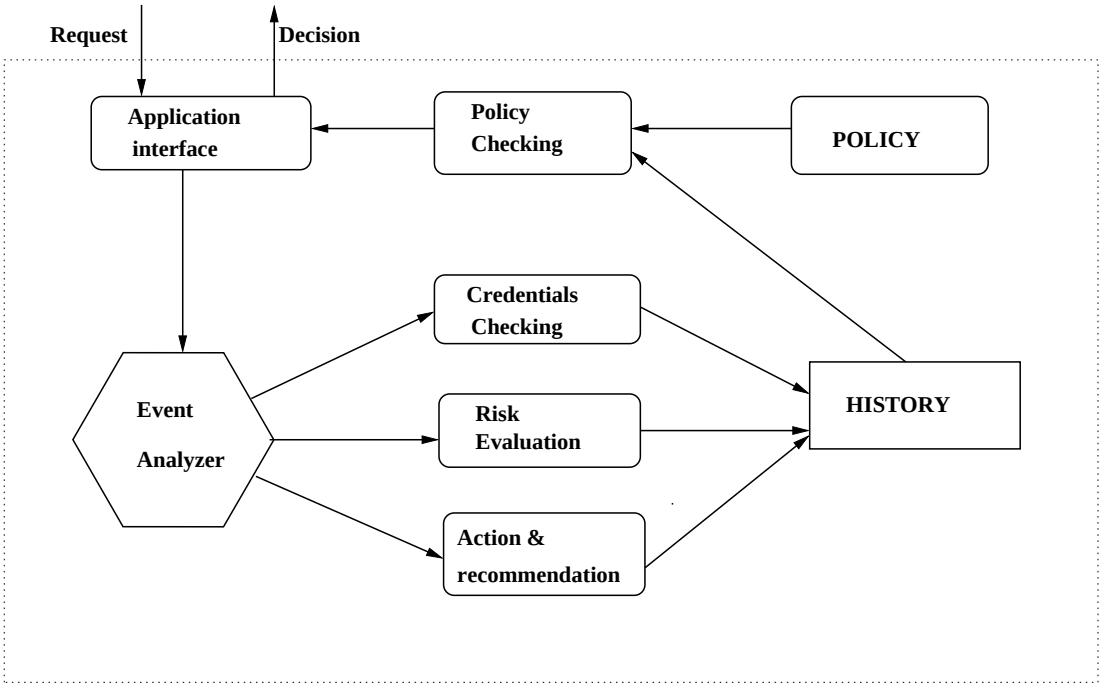


FIG. 5.1 – Système de gestion de la confiance

la politique définie dans *Policy*. Le résultat de cette vérification nous indique si la transaction avec ce partenaire peut être poursuivie.

5.2 Formalisation de la confiance

Cette section présente la formalisation de notre système. Nous expliquons comment la confiance est modélisée et comment une décision est prise pour établir une relation de confiance entre les entités du système.

5.2.1 Formalisation des événements

Les événenements utilisés dans le système sont prédéfinis et nous avons besoin d'un mécanisme pour les formaliser. Comme nous l'avons vu, toutes les actions et toutes les données échangées par le système sont observées et transformées en événements. Il est donc indispensable de pouvoir les identifier. Dans le cadre d'une application particulière, nous devons décrire l'ensemble des événements qui sont utiles.

Bien entendu, chaque application peut interpréter ses observations comme elle le désire, mais nous avons identifié un certain nombre de données qui apparaissent de manière récurrente et nous proposons de les intégrer de manière standard dans notre modèle. Nous avons retenu les informations concernant les qualifications, les recommandations, l'expérience et les risques ; elles sont modélisées directement par les événements correspondants.

Toutes les interactions observées par le système sont traitées par les composants *Application Interface* et *Event Analyzer* qui se chargent de les transformer en événements correspondants. Comme les événements sont organisés et contraints par une structure

d'événements, s'ils ne sont pas indépendants, ils doivent respecter les relations de causalité et ne pas provoquer de conflits.

Les événements sont à la base de la construction de l'historique et la politique. Ils sont considérés comme des atomes des prédicats de la politique. Lors d'une transaction, les événements sont sauvegardés dans une session de l'historique en respectant les contraintes induites par la structure d'événements définie pour l'application.

Nous détaillons ci-dessous les différents types pour la formalisation des événements en classe différentes.

Actions : Toutes les actions observées doivent respecter les relations définies dans la structure d'événements.

Les actions que nous pouvons modéliser comme des événements sont des demandes de services, des propositions d'échanges, des contre-propositions ou des confirmations de la demande...

Par exemple, dans le cas du scénario d'une transaction d'e-commerce entre un client et un vendeur, le client peut observer les actions suivantes :

- *passer_commande* : l'action de passer sa commande auprès du site marchand,
- *paiement* : le client paie les produits demandées au site marchand,
- *ignore* : le client ignore la demande de paiement.

Le vendeur, de son côté, peut observer les actions suivantes :

- *order_request* : l'événement indiquant que le client a passé une commande au marchand,
- *confirmer_commande_paiement* : le site confirme la commande et demande d'un paiement.

Notons que nous ne nous intéressons qu'aux actions-clés de l'application pour la formalisation des événements, celles qui peuvent concerner l'évaluation de la transaction.

D'autres événements sont construits à partir des *qualifications*, des *recommandations* ou du *risque* échangés pendant la transaction. Pour quantifier ou qualifier des données, nous utilisons comme moyen de transformation un pré-traitement qui est soit un calcul, soit une vérification.

Qualification : les qualifications (*credentials*) indiquent quelles sont les compétences ou les actions autorisées au propriétaire. La validité de ces qualifications est vérifiée par le module *Credential_Verification* dès qu'elles sont reçues d'un partenaire. Ce module génère des événements en indiquant leur validité ; par exemple : *valid_cert, invalid_cert, unknown_cert.*

Risque : l'entité évalue le risque sur la transaction en utilisant le mécanisme qui lui semble le plus adapté, comme par exemple le calcul du coût des conséquences de l'action entreprise. Cette tâche est assurée par le module *Risk_Evaluation.* Ce module génère des événements concernant le niveau de risque : *high_risk, low_risk, medium_risk.*

Recommandation : les recommandations des autres acteurs du système peuvent également être observées en tant qu'événements. L'entité peut définir et prendre en compte dans sa politique des événements concernant la recommandation (par exemple :*positive_recommendation, negative_recommendation, neutre_recommendation*).

Nous pouvons constater que ces types d'événements sont utilisables dans des applications différentes. Par exemple, pour les applications qui utilisent des *qualifications*, nous pouvons définir un ensemble d'événements dédié à cet usage : *valid_cert, invalid_cert, unknown_cert.*

5.2.2 Structure des événements

Les événements que nous avons formalisés doivent respecter la structure d'événements (relations de conflit et de causalité). C'est grâce à cette structure d'événements que nous pouvons définir nos politiques et construire l'historique des événements observés.

La structure d'évènement doit obligatoirement être adaptée à l'application qui l'utilise.

5.2.3 Historique

Les événements de toutes les transactions passées sont sauvegardés dans l'historique. L'historique est composé de plusieurs sessions où chaque session représente une transaction. Une session est un ensemble d'événements, qui est une *configuration* telle que présentée au chapitre 4.

5.2.4 Politique de confiance

L'utilisation de la structure des événements et de la logique PP-LTL présentée au chapitre précédent, nous permet de spécifier la politique de confiance de l'entité sous la forme d'une formule logique. Cette formule présente les conditions sous lesquelles l'entité fournit l'accès à ses ressources ou accède à des ressources.

L'intérêt pour chaque entité de disposer de sa propre politique est de pouvoir protéger ses ressources et sa sécurité de la manière qui lui semble la plus adaptée.

Pour protéger une ressource, l'entité peut placer différentes contraintes concernant la ressource telles que : des droits d'accès à la ressource, des contraintes d'intégrité, etc. La politique exprime toutes ces contraintes.

- *Contrôle de l'accès aux ressources*
  Pour protéger ses ressources, l'entité ne fournit pas d'accès sans discernement. Le droit d'accéder n'est fourni qu'aux entités qui satisfont les conditions requises par l'entité. Par exemple, un utilisateur peut accéder à la base des articles de IEEE s'il est abonné au service et ce droit d'accès ne lui sera concédé qu'après son authentification par *login/password*
- *Accès au service*
  Pour se protéger, l'entité n'est pas prête à accéder à un service ou à répondre à une proposition dans n'importe quelles conditions ; elle va chercher à minimiser les risques qu'elle prend. Par exemple, un utilisateur peut adopter une politique qui indique qu'il n'accède jamais à un site dont l'adresse appartient à une certaine liste noire. Il considère qu'accéder à un tel site est prendre un risque trop important de recevoir des *malwares* ou des virus.
- *Contraintes d'intégrité*
  Nous pouvons imposer à notre politique de respecter des contraintes d'intégrité sur les ressources. Par exemple, indiquer qu'il n'est pas possible d'ouvrir un fichier tant que celui-ci n'a pas été créé.

Après avoir réalisé l'analyse de toutes ces contraintes, il est possible à chaque entité de proposer sa propre politique en l'exprimant en logique PP-LTL.

5.2.5 Décision de la confiance

Le module de vérification de la politique est basé sur le *dynamic model checking* [59]. L'objectif de ce module est de vérifier la conformité de la politique et de l'historique. Chaque nouvel événement observé lors d'une transaction est ajouté à son historique et le module vérifie que la politique est toujours satisfaite.

5.3 Scénario 1 : Transaction dans le cadre du commerce électronique

Dans cette section, nous étudions le scénario d'une application de commerce électronique et nous proposons un modèle basé sur notre système de gestion de la confiance pour en assurer la sécurité des transactions. Notons que ce scénario a été présenté au chapitre 4.

5.3.1 Problème de confiance lors d'une transaction

Dans cet exemple nous considérons une transaction entre un marchand (le *seller*) et un client (le *buyer*). Le client veut acheter un produit chez le marchand. Le marchand observe la demande du client, et peut prendre en considération différents paramètres concernant la transaction : le profil du client, le montant de la transaction... Pour que la commande soit validée, le client doit payer son achat au marchand et pour ce faire il dispose de plusieurs méthodes : paiement par carte bancaire, virement, chèque ou mandat. Quand le marchand dispose de toutes les informations concernant la commande du client, comment peut-il décider de valider ou pas la transaction ? Le marchand dispose d'un ensemble d'informations concernant les transactions passées de ce client (sauvegardées dans son historique), le montant de la transaction courante, les informations relatives au paiement... Pour réduire les risques, le marchand met en place une politique de confiance qui va l'aider à prendre une décision sur la suite à donner à la commande que le client vient de passer. Si l'historique (avec ses configurations courantes) satisfait sa politique de confiance, la transaction peut se poursuivre, sinon elle sera interrompue. Ce scénario est illustré figure 5.2.

Dans cette figure, le client envoie sa *commande* au marchand et le paie en utilisant une des méthodes proposées par le marchand. Après avoir eu toutes ces informations, le marchand prend une décision pour traiter la demande du client. Cette décision est prise par le module de *Gestion de la confiance.*

Nous pouvons distinguer trois possibilités sur l'état d'un paiement : *paiement valide*, *paiement invalide* et *paiement inconnu* :

- *paiement valide* : le marchand a reçu du client un paiement effectué par un moyen de *confiance*, par exemple un paiement par carte bancaire, par un mandat (le montant de la transaction a été effectivement crédité sur son compte).
- *paiement invalide* : le marchand reçu du client une preuve de paiement invalide (carte bancaire volée, fausse monnaie électronique).
- *paiement inconnu* : le marchand a reçu un paiement mais il n'a pas encore pu en vérifier la validité. Par exemple, le marchand a reçu un paiement par chèque mais il ne l'a pas encore encaissé.

À la fin de chaque transaction, le marchand peut évaluer le comportement du client concernant la transaction. Cette évalution peut être *posivitve*, *négative* ou *neutre.*

Pour minimiser les risques pris lors d'une transaction tout en simplifiant ses relations avec ses *bons clients*, le marchand peut mettre en œuvre la politique de confiance suivante :

- Si les risques sont faibles (le montant de la transaction est petit), il accepte de mener à leur terme les transactions avec des clients qu'il n'a jamais évalués de manière *négative* par le passé.
- Si les risques sont élevés (le montant de la transaction est important), il accepte de poursuivre la transaction s'il a reçu le paiement et si le client n'a jamais été évalué de manière *négative*.

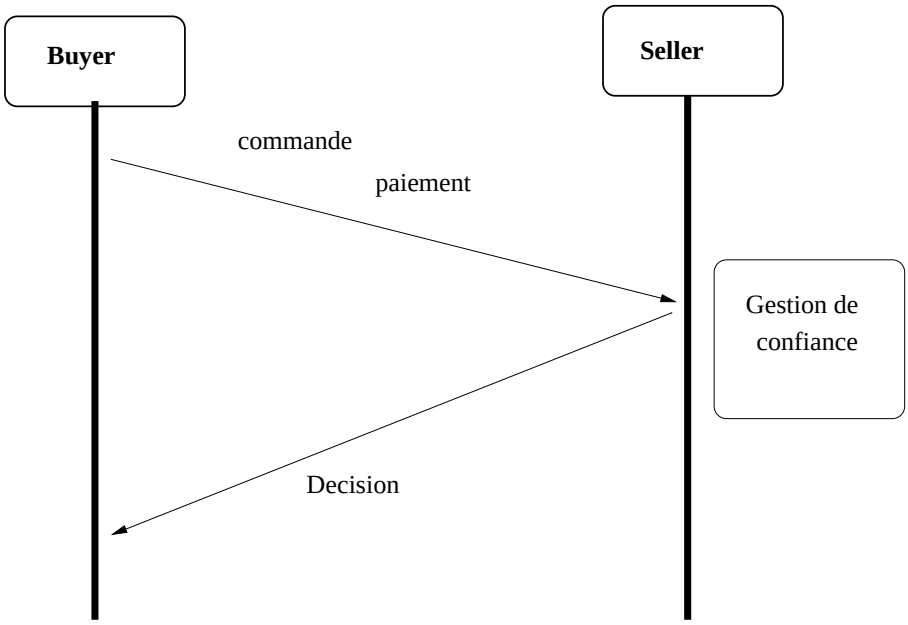


FIG. 5.2 – Transaction du commerce électronique

5.3.2 Modélisation du scénario

Nous proposons dans cette section une structure d'événements et une politique de confiance adaptées à ce scénario. Nous précisons également que cet exemple de modélisation a pour objectif de prendre une décision sur la transaction du côté du marchand. Nous présenterons au chapitre 6 une autre modélisation qui permettra au client et au marchand de mener à bien la transaction sous la forme d'une négociation.

Structure d'événements

La structure d'événements observée par le marchand est illustrée dans la figure 5.3. Les événements observés par le marchand se composent des *actions* et des *données* transmises par le client lors de la transaction.

- *valid_payment, unvalid_payment, unkown_payment* : le marchand a reçu un paiement qui est valide, invalide ou qui n'est pas encore vérifié.
- *positive, negative, neutre* : de quelle manière le marchand évalue le client.
- *high_risk, low_risk, average_risk* : niveau de risque de la transaction.
- *object-order* : le marchand a observé que le client envoie une commande.

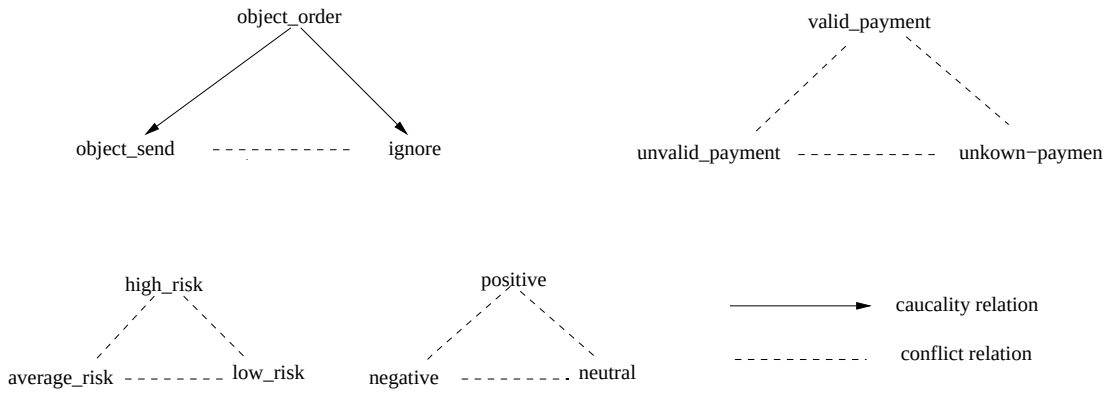


FIG. 5.3 – Structure d'événements observée par le seller

– *object-send* : le marchand a validé la transaction et envoie la marchandise au client.
– *ignore* : le marchand ne valide pas la transaction.

Notons que le risque sera calculé par un modèle de risque que nous présentons au chapitre 7. Pour l'instant, nous considérons que le risque associé à la transaction est représenté par un événement qui peut prendre comme valeur *high_ risk*, *low_ risk* ou *average_ risk*.

Politique de confiance

Après avoir identifié des séquences à risque, le marchand peut proposer les politiques suivantes pour en limiter les conséquences :

– Le marchand rejette immédiatement la commande du client dont le paiement est invalide, ou si toutes les évaluations des transactions antérieures sont négatives :

$$(\psi_1) : ignore \rightarrow (object\_order) \wedge (unvalid\_payment \vee G^{-1}(negative))$$

– En cas de risque important, le marchand n'accepte que les commandes de client dont le paiement est valide ou si toutes les évaluations de transactions antérieures sont positives :

$$(\psi_2) : high\_risk \rightarrow (valid\_payment \vee G^{-1}(positive))$$

– En cas de risque faible, le vendeur accepte les commandes du client dont le statut du paiement est encore inconnu même si par le passé il a eu des évaluations négatives :

$$(\psi_3) : low\_risk \wedge unkown\_payment \wedge F^{-1}(negative)$$

La politique globale de confiance est une fonction (disjonction) des sous-politiques que nous venons de présenter :

$$\psi_{global} \equiv \psi_1 \wedge \psi_2 \wedge \psi_3$$

Historique et vérifications

À chaque instant de la transaction, le marchand collecte les événements concernant la transaction et les sauvegardes dans la session courante de l'historique. L'ensemble des événements observé lors d'une transaction entre un client et un marchand permet de définir une session de l'historique. Le marchand peut avoir un nombre quelconque de sessions dans son historique.

Supposons que le marchand ait l'historique suivant pour un client :

$$\begin{aligned} H_V = &\{object\_order, low\_risk, unkown\_payment, negative\} \\ &\{object\_order, high\_risk, valide\_payment, positive\} \\ &\{object\_order, low\_risk, valide\_payment, positive\} \\ &\{object\_order, low\_risk, valide\_payment, positive\} \end{aligned}$$

Si la transaction courante avec ce client, la session x_i contient les événements suivants :

1. $\{object\_order, low\_risk, unkown\_payment\}$
   Nous voyons que H_V et x_i satisfont la politque ψ_{global}, cette commande est donc validée.
2. $\{object\_order, high\_risk, unkown\_payment\}$
   Dans ce cas, nous voyons que x_i ne satisfait pas la politique (et donc l'historique), cette commande n'est pas validée.

5.3.3 Discussion

Dans ce scénario, nous avons présenté la mise œuvre de la politique de confiance du seul point de vue du marchand, mais nous devons considérer la politique de confiance des deux parties en présence dans la transaction, le marchand et le client. C'est de l'interaction de ces deux politiques de confiance que naitra la négociation entre les deux parties, qui permettra d'obtenir une confiance mutuelle pour la transaction. Ce problème sera étudié au chapitre 6 avec l'introduction de notre système de négociation de la confiance.

5.4 Scénario 2 : Évaluation de la contribution d'un utilisateur à Wikipédia

Dans cette section nous présentons une structure d'événements et une politique de confiance dont l'objectif est d'aider à l'évaluation des contributions des utilisteurs aux articles de Wikipédia. Un des principaux problème de Wikipédia est de maintenir un niveau de qualité suffisant aux articles de l'encyclopédie et donc de savoir comment faire confiance aux contributions des utilisateurs.

Le site web Wikipédia est une encyclopédie qui évolue selon un modèle collaboratif. Tout le monde peut y créer ou modifier des articles pour en améliorer les contenus et y faire partager ses connaissances. Wikipédia a une audience mondiale, elle dispose d'un très grand nombre[2] d'articles consultés et modifiés par un très grand nombre d'utilisateurs ; la question de la gestion de leur intégrité, de leur qualité est très importante pour continuer à assurer le succès de l'encyclopédie.

[2] plus d'un million d'articles en français à la date du 1er octobre 2010
http://fr.wikipedia.org/wiki/Wikipédia:Statistiques

Chaque utilisateur qui modifie ou publie un article de Wikipédia est identifié soit par son identifiant s'il dispose d'un compte déjà enregistré, soit par son adresse IP. Tous les utilisateurs pouvant modifier un article, on peut donc obtenir deux résulats : une *contribution* si la modification proposée est utile à l'article ou un *dommage* si cette modification introduit des erreurs ou est un acte de vandalisme. Un utilisateur peut donc détériorer un article, mais quelqu'un d'autre peut contribuer à le corriger et ainsi de suite. Un article sur Wikipédia peut donc être vu comme une succession de contributions qui peuvent être positives ou négatives.

Les utilisateurs de Wikipédia sont tous égaux, mais certain d'entre eux se sont spécialisés dans la vérification des modifications récentes : les patrouilleurs. Leur activité consiste à déterminer si une modification est un acte de vandalisme et si c'est le cas, à annuler au plus tôt les modifications apportées à l'article.

Chaque article a un historique qui est composé de la liste des modifications, où à chaque modification sont associés la date, l'identifiant de l'auteur, le type (discussion ou contribution) et le moyen de consulter et de *défaire* cette modification.

À chaque utilisateur est associé l'historique de ses contributions. Chaque élément de cette liste contient la date de modification, le nom de l'article modifié et en quoi consistait la modification, sous la forme de la différence entre la version antérieure et la version modifiée.

Les données d'historique concernant l'utilisateur et l'article constituent une bonne source d'informations pour les analyses du patrouilleur sur les modifiations apportées. Il peut déterminer un niveau de réputation de l'utilisateur en se basant sur le nombre de contributions utiles qu'il a apportées. L'historique de l'article donne des informations sur l'ensemble des comportements de l'utilisateur. En combinant ces résultats avec sa politique et son expérience (historique personnel), il peut prendre une décision sur l'action de l'utilisateur.

Il semble donc que tout mécanisme soit le bienvenu pour aider les patrouilleurs dans leur tâche. Un système de gestion de la confiance dédié à l'observation des modifications permettrait aux patrouilleurs de décider de l'annulation ou non d'une modification d'un utilisateur sur un article. Cette aide pourrait tirer parti des historiques associés aux utilisateurs et aux documents mais aussi de l'expérience du patrouilleur.

5.4.1 Modélisation

Nous considérons les interactions entre deux entités : un *utilisateur* qui a effectué une modification sur un article et un *patrouilleur* qui cherche à identifier des actions de vandalisme.

Une session complète concernant la modification d'un article peut se dérouler de la manière suivante :

- l'utilisateur effectue une *modification* sur un article ;
- le patrouilleur analyse la modification en utisant ses données d'historique concernant cet utilisateur ;
- le patrouilleur *annule* la modification s'il détecte un acte de vandalisme.

Le patrouilleur utilise l'identifiant de l'utilisateur et son historique des modifications pour déterminer s'il a déjà effectué des actes de vandalisme.

Le patrouilleur peut annuler des modifcations de l'utilisateur dont l'adresse IP est dans une liste noire, ou dont le compte est considéré comme appartenant à un *spammer* dans la communauté Wikipédia.

En utilisant l'historique des modifications d'un utilisateur, le patrouilleur peut estimer une valeur de réputation (en calculant le pourcentage de contribution considérées comme pertinentes par exemple). Cette valeur permettra de lui accorder un niveau de réputation : *elevé*, *moyenne* ou *faible*.

Le patrouilleur peut aussi prendre en compte les modifications que l'utilisateur a apporté à l'article pour savoir si la contribution est positive (par exemple, il doit avoir fait au moins une modification pertinente).

5.4.2 Structure d'événements

Un contributeur de Wikipédia est identifié par son adresse IP ou par un nom. Nous définissons les événements suivants :

- *auth, anonym* : l'utilisateur est authentifié ou anonyme (adresse IP) ;
- *modif* : il s'agit d'une modification d'un utilisateur ;
- *revoque, ignore* : le patrouilleur annule (ou non) la modification. Une annulation est aussi une modification ;
- *high_rep, medium_rep, low_rep* : réputation de l'utilisateur, respectivement bonne, moyenne ou mauvaise, pour ses activités sur Wikipédia. Ces valeurs de réputation sont calculées en utilisant l'historique de ses activités.

La structure d'événements observée par le patrouilleur est présentée figure 5.4.

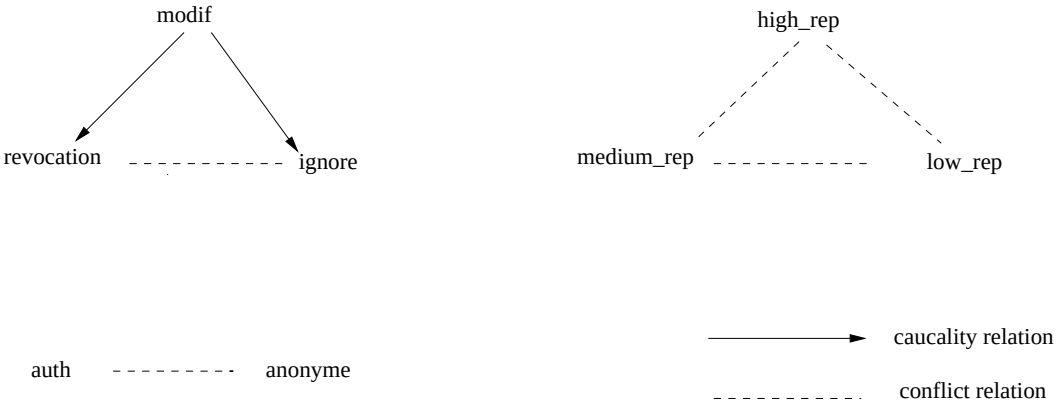


FIG. 5.4 – Structure d'événements d'un patrouilleur

5.4.3 Politique

Pour porter un jugement sur l'activité de modification d'un utilisateur, le patrouilleur peut proposer des politiques. Le but est de supprimer des modifications qui sont susceptibles d'être du vandalisme.

- Annuler immédiatemet la modification d'un utilisateur qui n'est pas authentifié et dont la réputation est mauvaise ou dont toutes les modification précédentes sur cet article ont été annulées :

$$\psi_1 : [G^{-1}(modif \wedge revoque) \vee (anonym \wedge low\_rep)] \rightarrow revoque$$

- La contribution est acceptée (le patrouilleur ne l'annule pas) si l'utilisateur est authentifié, s'il a déjà effectué des modifications pertinentes sur l'article ou si sa

réputation n'est pas mauvaise :

$$\psi_2 : (auth \vee F^{-1}(modif) \vee \neg low\_rep) \rightarrow ignore$$

La politique globale du patrouilleur est $\psi_P \equiv \psi_1 \wedge \psi_2$

5.4.4 Historique et vérification

Voici quelques exemples d'historiques d'un patrouilleur concernant un utilisateur particulier. Ces informations sont utilisées par l'algorithme qui vérifie la satisfaction de la politique et dont le résultat permet au patrouilleur de prendre la décision d'annuler ou non une contribution.

Supposons que le patrouilleur ait l'historique suivant pour un utilisateur :

$$\begin{aligned} H_P = &\{modif, anonym, low\_rep\, revoque\} \\ &\{modif, auth, low\_rep, ignore\} \\ &\{modif, anonym, high\_rep, ignore\} \\ &\{modif, auth, high\_rep, ignore\} \end{aligned}$$

Considérons que chaque modification que l'utilisateur apporte à l'article soit considérée comme une session. Soit x_i la session courante :

1. $x_i = \{modif, auth, medium\_rep\}$. L'utilisation de la politique ψ_P et de l'historique $(H_P.x_i)$ nous permet de déterminer que l'événement *ignore* est un candidat satisfaisant. Le patrouilleur n'annule pas cette modification.
2. $x_2 = \{modif, anonym, low\_rep\}$. L'événement *revoque* satisfait la politique. La modification est annulée.

5.4.5 Discussion

Dans ce scénario, nous nous sommes seulement intéressés à déterminer si la contribution d'un utilisateur était succeptible d'être un acte de vandalisme. Les données que nous utilisons sont uniquement l'historique des modifications de l'utilisateur et son identifiant.

5.5 Synthèse

Ce chapitre présente notre modèle de gestion de la confiance et l'utilisation qui est faite du modèle de la structure d'événements. Nous y présentons les éléments principaux : la formalisation des événements, la structure d'événements, la spécification de la politique et le modèle de vérification.

Dans la deuxième partie du chapitre, nous avons présenté deux exemples d'applications de gestion de la confiance qui utilisent notre approche : un système de commerce en ligne et le module de gestion de la confiance pour l'encyclopédie Wikipédia.

Chapitre 6

Négociation de la confiance

Dans ce chapitre, nous présentons notre modèle de la négociation de la confiance, puis nous illustrons son utilisation avec un exemple. Notre modèle de négociation utilise le modèle de la confiance que nous avons présenté au chapitre 5 et la négociation de la confiance prend la forme d'un ensemble d'interactions entre les participants, et se traduit sous la forme d'un protocole engendré dynamiquement. Le scénario de l'application prise en exemple est l'établissement de la confiance lors d'une transaction entre deux acteurs d'un site marchand en ligne : le vendeur et un acheteur. Le contenu de ce chapitre détaille les travaux que nous avons présentés dans [47].

6.1 La notion de négociation

La notion de négociation telle qu'elle est utilisée en informatique et surtout dans la communauté des réseaux et de la sécurité a un sens que l'on peut considérer comme assez éloigné du sens commun que l'on donne à ce mot.

> **négocier**
> *verbe transitif (latin negotiari, faire du commerce)*
> Discuter de quelque chose avec quelqu'un en vue de l'établir, de l'obtenir : Négocier la paix avec l'ennemi.
>
> Dictionnaire Larousse.

Voici un exemple pour illustrer notre propos : un propriétaire veut vendre sa maison, il l'affiche à un certain prix (plutôt élevé) tout en sachant qu'il est prêt à baisser ce prix jusqu'à une certaine valeur plancher qu'il s'est fixée. Un client veut faire l'acquisition de ce bien ; il fait une offre pour un montant inférieur au prix demandé tout en sachant qu'il est prêt à offrir plus si nécessaire. Nous avons donc des politiques (vendre le plus cher possible, acheter le moins cher possible) propres à chaque participant, chacun a un objectif à maximiser et personne ne sait jusqu'où l'autre est prêt à aller. Si au cours des échanges on peut finaliser la transaction (le client fait une offre acceptée par le vendeur ou accepte sa proposition), personne n'a eu besoin de revenir sur les conditions imposées par sa politique et il n'y a pas réellement eu de négociation ; on peut considérer que l'on a déroulé un algorithme et qu'il a fourni une solution acceptable pour les deux parties. Par contre, si aucune solution n'est trouvée, on a atteint un point de blocage et la seule solution pour pouvoir conclure la transaction positivement conduit les participants à modifier leur politique, à aller au delà de ce qu'ils avaient prévu ; on est dans une réelle

phase de négociation. Si personne ne modifie sa position ou sa politique, la transaction ne peut avoir lieu et la négociation est donc un échec.

Dans les communautés des réseaux et de la sécurité, on parle de négociation dès lors qu'une alternative est offerte dans le déroulement des protocoles. Par exemple dans le cas du protocole HTTP[31], le *content negociation* a pour objet d'adapter au mieux le format des ressources réclamées par le client : le client exprime ses souhaits au travers du header *Accept*, le serveur renvoie le document dont il dispose qui maximise cette formule.

```
Accept: text/html; q=1.0,text/*; q=0.8,image/gif; q=0.6,image/jpeg;
        q=0.6,image/*; q=0.5,*/*; q=0.1
```

Le protocole IKE[39] permet de négocier les paramètres d'une connexion IPSEC[29], son objectif est de déterminer les algorithmes d'authentification, de chiffrement et de scellement des données acceptables pour les deux parties. Cette phase de négociation a été définie pour permettre l'ajout de nouveaux algorithmes, et de cette manière rendre le protocole indépendant des techonologies disponibles à un instant donné. Malgré tout, la négociation se résume à la découverte des méthodes acceptées par les deux parties. Par exemple, pour ce qui est des méthodes de chiffrement et de scellement des données, chacun définit dans sa configuration ce qui est acceptable à son sens. L'initiateur de la connexion fait des propositions à son correspondant jusqu'à obtenir une réponse positive de sa part.

Comme nous venons de le voir au travers de ces deux exemples, l'utilisation du mot « négociation » est quelque peu surfait, il n'est jamais question de modifier une quelconque politique en cas d'échec : dans le cas du protocole HTTP, le serveur renverra ce qu'il a, n'importe quoi faisant l'affaire (`q=0.5, */* ; q=0.1`) et dans le cas de IKE/IPSEC, la communication ne sera pas établie.

6.2 Système de négociation

Nous présentons dans cette section les bases pour la conception de notre système de confiance. Nous abordons d'abord le sens du terme *négociation* utilisé dans notre système. Ensuite, nous parlons de l'approche *événement* pour notre système de négociation.

6.2.1 Terme de la négociation de confiance

Comme nous l'avons vu dans l'état de l'art, le terme *« négociation de la confiance »* a des sens différents dans les systèmes que nous avons étudiées et la *« négociation »* n'a parfois pas le sens commun que nous lui donnons. Nous précisons dans quel sens nous avons utilisé ce terme dans notre système.

Une négociation se résume donc en général à une transaction entre deux acteurs dont l'objectif est de leur permettre d'obtenir simultanément un niveau de confiance suffisant. Les systèmes existants tels que Trust-X [5, 6], TrustBuilder [26, 84] sont bien dans ce cadre là, ils voient la négociation comme un échange d'informations préalable entre deux parties pour établir une confiance mutuelle et réaliser des transactions dans la suite.

Dans notre approche de système de négociation, le terme de *« négociation de la confiance »* désigne les échanges entre deux parties qui satisfont les contraintes suivantes :

- Chaque partie engagée dans l'échange a son propre objectif. Le but de l'échange est que les deux parties puissent mener à bien l'échange tout en satisfaisant leurs objectifs.
- Chaque partie gère sa politique. La politique est un moyen pour protéger ses objectifs et ses ressources personnelles.
- À chaque étape de l'échange, lorsque l'on reçoit une *proposition* de son partenaire, on verifie que la politique de confiance est toujours satisfaite. Si c'est le cas, cela permet de déterminer l'action suivante à entreprendre et sinon cela permet de faire une nouvelle proposition à son partenaire. C'est ici que la notion de négociation prend tout son sens.
- La fin de l'échange est atteinte lorsque les deux parties ont satisfait leurs objectifs ou qu'elles arrivent à une impasse, la négociation est alors un échec.

Il nous faut préciser que les deux parties engagées dans la négociation ne se sont peut-être jamais rencontrées, mais que ce n'est pas forcément une obligation. Les connaissances qu'elles peuvent avoir l'une de l'autre couvrent les aspects suivants :

- elles ont peut-être déjà mené des transactions ensemble, leurs historiques contiennent alors des informations sur leurs comportements respectifs ;
- elles partagent le même but dans la transaction : celui de mener à bien la transaction et d'atteindre leur but personnel[1].

L'aspect le plus important est que personne ne connait la politique de son partenaire ; on ne sait donc pas, à un instant donné, comment il va réagir. Par contre, comme nous imaginons que leur objectif est de mener à bien la transaction, cela permet d'identifier les réactions possibles à une action et c'est une aide importante pour la modélisation du système.

6.2.2 Architecture du système de négociation

Nous considérons la négociation comme un processus d'échange d'informations (des propositions ou des réactions à des propositions) entre les participants et de décisons prises indépendamment par chacun de ces participants. Chaque acteur observe la transaction sous la forme d'une succession d'*événements*.

À chaque étape, lorsqu'un acteur reçoit un message de son partenaire, la proposition est analysée et traduite en un événement utilisable par la politique. L'ensemble de ces analyses est effectué par le module de gestion des événements. Les réactions aux diverses actions dépendent de la stratégie de l'acteur, qui est confiée au processus de *négociation de la stratégie*.

Nous modélisons notre système de négociation dans le cas ou deux parties sont en négociation. Chaque partie engagée dans la négociation est associée à *un agent de négociation*. La Figure 6.1 présente l'architecture générale de cet agent.

6.2.2.1 Composants

L'agent de négociation que nous présentons se décompose en deux parties principales : la gestion du *protocole et de l'interface* et la gestion de la *négociation de la confiance*.

[1]Bien que le but personnel de chaque partie soit différent (obtenir le paiement du bien livré ou recevoir ce qui a été acheté), il s'agit bien de le satisfaire.

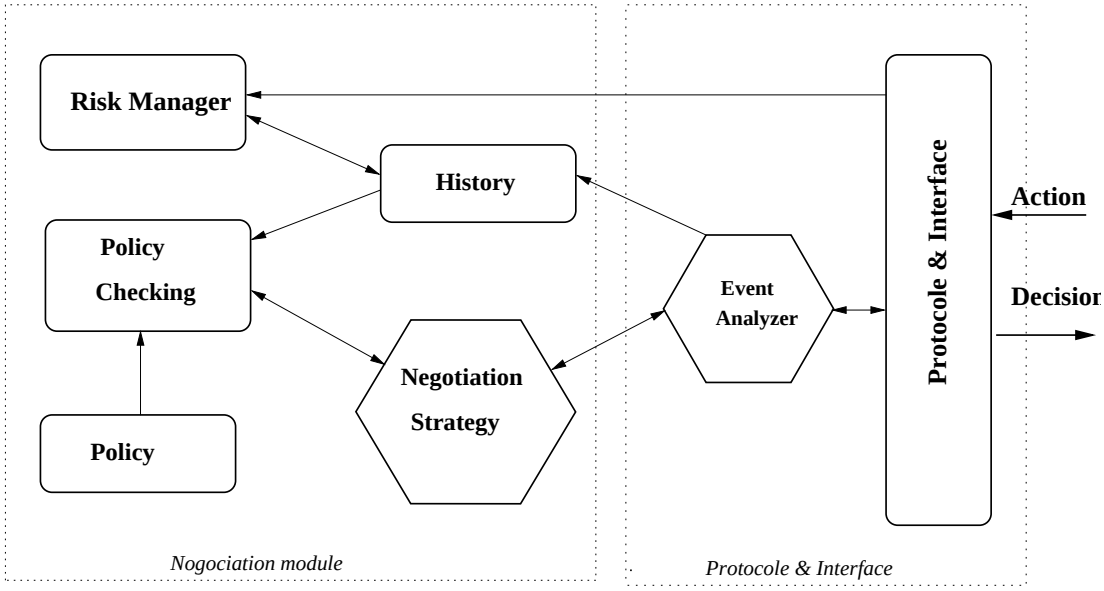


FIG. 6.1 – Architecture du système de négociation de confiance

Protocole et interface

La partie *protocole et interface* gère la transformation des messages émis ou reçus sur le réseau en événements et vice-versa. Elle est composée d'un élement chargé de la gestion des communications et de la mise en forme protocolaire des messages et d'un élément, le *Event Analyzer*, qui interprète ces messages pour les transformer en événements. Le *Event Analyzer* se comporte comme une interface entre *le protocole* et le module *négociation*. Nous effectuons une présentation détaillée de ce composant dans la section 3.2. Nous y abordons les différents aspects du protocole (les échanges, les message...) et les fonctions du *Event Analyzer*.

Module de négociation

Le module de négociation prend en charge toutes les fonctionnalités concernant les décisions de confiance et la stratégie de négociation qui contrôle les échanges. Pour remplir cette tâche, ce module utilise la structure d'événements. Ce module a deux sous-composants : le composant de *gestion de la confiance* et le composant *Stratégie de négociation*.

- La *gestion de la confiance* est chargée de prendre les décisions sur la confiance à accorder à l'action demandée. Le modèle de gestion de confiance que nous utilisons est celui que nous avons présenté au chapitre 5. Nous rappelons brièvement les points importants : les observations représentent le comportement des acteurs, ces observations sont transformées en événements, ces événements sont conservés sous la forme de sessions dans un historique, chaque acteur dispose de sa politique de confiance et à chaque instant il lui est possible de vérifier que son historique satisfait toujours cette politique. Le résultat de cette vérification permet de prendre une décision quant à la suite à donner à la transaction : *avons-nous toujours confiance*.
- La *stratégie de négociation* est chargée de contrôler les échanges entre les entités

(quelle est l'action à engager sur une proposition). Nous présentons en détail ce composant dans la section 6.4.7

6.2.2.2 Fonctionnement

Le fonctionnement en général de notre système de négociation peut être spécifié par les échanges entre les deux entités en négociation. La figure 6.2 illustre les interactions entre A et B.

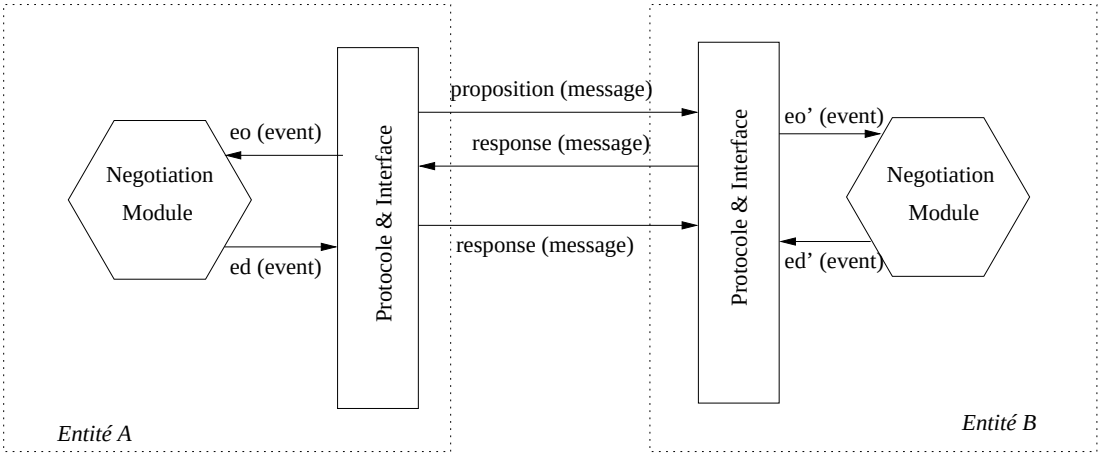


FIG. 6.2 – Architecture de négociation : Fonctionnement

- A veut faire une transaction avec B, il envoie à B une proposition sous la forme d'un message. A conserve l'événement représentant sa proposition dans le module de négociation.
- B reçoit le message de A, ce message sera transformé en un événement représentant une proposition. Le module de négociation de B sélectionne un événement pour répondre à la proposition de A. Le module de gestion de la confiance vérifie que la politique de B est toujours satisfaite puis envoie le message correspondant à A.
- A reçoit la réponse de B et réagit de la même manière pour répondre à B.
- Les échanges entre A et B continuent jusqu'à la fin de la transaction ; que les deux entités atteignent leurs objectifs ou que la négociation soit un échec.

6.3 Protocole de négociation

Dans notre approche, le protocole de négociation est une séquence d'actions des deux parties. Il est constitué d'une suite d'échanges de messages. Ces messages sont observés et traduits par chacune des parties sous la forme d'événements, et ils doivent en respecter la structure. Autrement dit, l'ordre dans lequel les messages sont échangés entre les deux parties est induit par leurs structures d'événements.

6.3.1 Messages

Pour les messages échangés dans le protocole, nous nous intéressons à leurs deux aspects : leur type et leur structure.

Types de messages

Selon la nature des informations à échanger, nous pouvons classer les messages en quatre catégories élémentaires :

- *Proposition* : ce sont les messages qui indiquent une proposition d'échange, une demande de service. Quand une partie envoie une message de *proposition*, elle s'attend à recevoir un message de type *proposition* ou de type *Rejet* de son partenaire.
- *Négociation* : ce message demande au partenaire de modifier des données échangées à l'étape précédente ou de faire une autre proposition. Lorsque l'on reçoit ce type de message, si l'on est d'accord ou si c'est possible, on répond par une nouvelle proposition, sinon un message de type *Rejet* est envoyé.
- *Rejet* : les messages de ce type indiquent que l'on n'accepte pas la *proposition* ou la demande de *négociation*. Lorsqu'elle reçoit ce message, l'entité met fin de la transaction.
- *Conclusion* : Le message indique la fin de la transaction. En recevant un message du type *Proposition* ou du type *Rejet*, on peut mettre fin à la négociation en utilisant ce message. La négociation est une réussite si son objectif a été atteint, un échec sinon.

Nous distinguons ces types de message pour faciliter les échanges entre les deux entités les modélisant par la structure d'événements. Les messages sont constitués à partir des *événements* observés. Nous présentons dans la section suivante la procédure de transformation *événement* → *message* et le format standard des différents types de message.

Structure du message

Le format des messages peut être adapté à chaque application, mais il est préférable d'utiliser un format standard. Certains messages peuvent contenir des données sensibles et il est important de les prendre en compte lors de leur définition. Les messages doivent contenir les informations suivantes : l'identifiant de l'entité partenaire, le type du message et les données échangées.

Un message est constitué des éléments suivants :

```
structure message{
    Identifiant: entite_msg;
    Type: type_msg;
    Data: donnee_msg;
}
```

Le champ `entite_msg` identifie l'entité partenaire de la négociation. Le champ `type_msg` décrit le type de message, et il peut être d'un des types : *Proposition*, *Négociation*, *Rejet* ou *Conclusion*. Le champ `donnee_msg` décrit des données échangées.

6.3.2 Échanges

Le déroulement complet du protocole est conservé sous la forme d'un ensemble d'événements dans une session de l'historique. Ce déroulement est spécifié par la suite des échanges entre les deux parties. Supposons que deux entités A et B décident de mener à bien une transaction, ils échangent des propositions sous la forme de messages

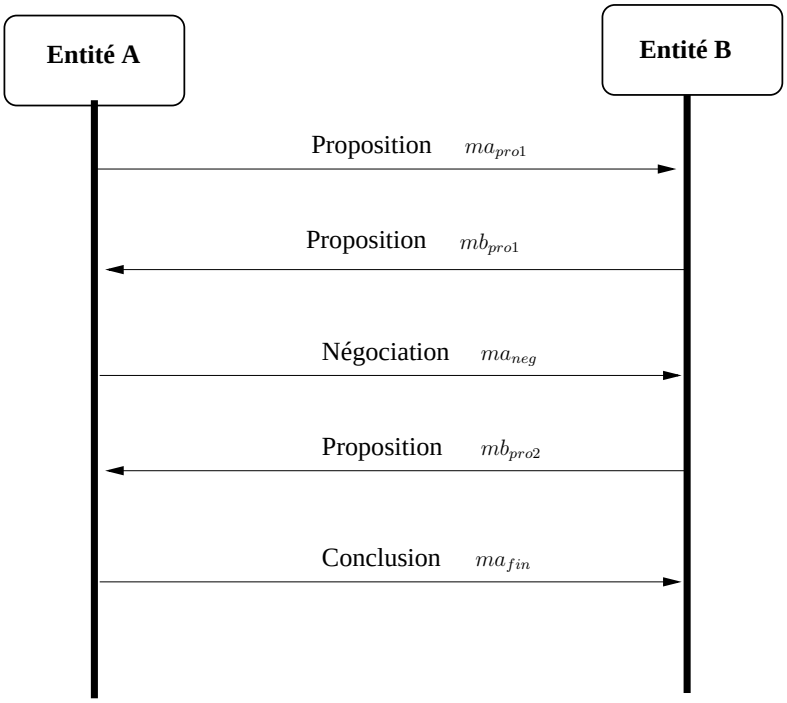


FIG. 6.3 – Un scénario de négociation

et ils peuvent *négocier* sur leur proposition. Le processus de négociation se déroule de la manière suivante :

- Le protocole démarre par une *proposition* de A à B en vue d'obtenir un service ou une ressource : ma_{pro1}.
- B analyse la demande de A et lui répond par une proposition d'échange mb_{pro1}. Cette proposition peut être une demande d'informations complémentaires (une qualification nécessaire pour avoir le droit d'accéder au service par exemple). Si B n'accepte pas cette proposition, il envoie à A un message de *conclusion* ou de *négociation*. Si c'est un message de conclusion, cela amène à la fin de la négociation.
- A reçoit à son tour un message de B, qui peut être une *conclusion*, une *proposition* ou une *négociation*, et l'analyse. Si c'est une *proposition*, A peut l'accepter, la refuser (avec le message de *Conclusion*), faire une autre proposition (avec la *Proposition*) ou faire une re-négociation (message de Négociation). Si c'est une *re-négociation*, A peut la *rejeter* (avec le message de *Rejet*) ou effectuer une autre *proposition*. Sinon, le message *Conclusion* signifie la fin de la négociation.
- Le processus d'échange des messages continue de cette manière jusqu'au moment où les deux parties atteignent la *fin de l'échange*, indiquée par le message *Conclusion*.

Tous les messages reçus sont analysés par l'*Event Analyzer* et interprétés sous la forme d'événements. Tous les messages envoyés sont constitués des événements fournis par le module *Strategy Negotiation*, et convertis en messages par l'*Event Analyzer*.

Dans le scénario illustré par la figure 6.3, l'entité A envoie une proposition initiale à B (ma_{pro1}) ; B lui répond par une contre-proposition (mb_{pro1}). A envoie alors à B une demande de négociation (ma_{neg}) ; Puis B fait une proposition alternative (mb_{pro2}).

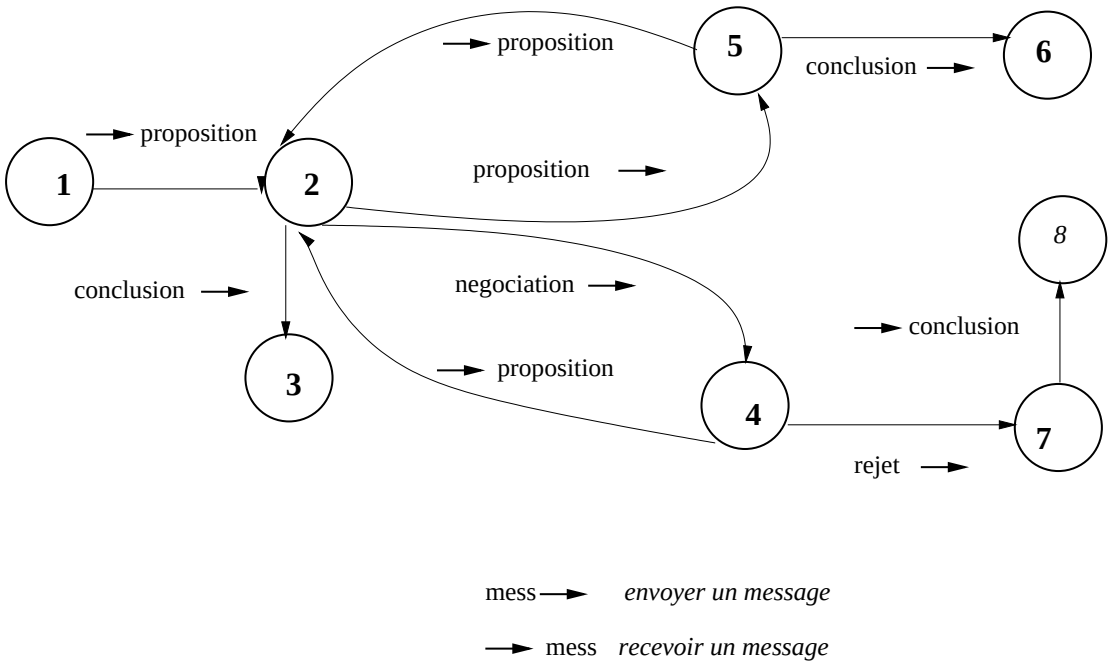


FIG. 6.4 – Automate de négociation

Finalement A accepte cette proposition et met fin à la négociation par le message de *conclusion* ma_{fin}.

Le schéma 6.4 présente les différentes étapes possibles lors de la négociation. Chaque étape est représentée par un état de l'automate. Chaque type d'échange entre les entités est spécifié par un changement d'état dans l'automate, la flèche indiquant si le message est envoyé ($mess \rightarrow$) ou reçu ($\rightarrow mess$).

Dans cette figure, l'état ① est l'état initial de l'entité. Quand il reçoit une proposition, l'automate passe à l'état ②. Ensuite, il peut passer aux états ③, ④ ou ⑤ en fonction de l'interprétation qu'il fait de la proposition reçue à l'étape précédente : mettre fin à la négociation (objectif atteint ou non), faire une contre-proposition ou faire une demande de re-négociation. En passant à l'état ⑥, l'entité indique qu'elle a atteint la fin de la transaction. L'état ⑦ indique que l'entité fait face à un refus de son partenaire (message reçu *rejet*). L'état ⑧ indique que l'entité envoie une conclusion après avoir refusé la négociation.

6.3.3 Fin de l'échange

En utilisant la structure d'événements pour modéliser notre système de négociation, nous voyons qu'une exécution du protocole est équivalente à une session de l'historique. Une session de l'historique qui représente l'exécution complète (lorsque la négociation est terminée) du protocole est une *configuration maximale* telle que nous l'avons présentée dans la section 4.5.2

Dans le processus de négociation, quand une partie se trouve dans une *configuration maximale*, elle en déduit qu'elle a atteint la fin des échanges du protocole, que le but de la négociation soit atteint ou pas.

En terme de messages échangés par le protocole, lorsqu'il y a envoi d'un message de type *conclusion*, cela indique la fin de la négociation.

Quand l'entité trouve un événement qui lui permet de compléter sa session sous la forme d'une configuration maximale cela peut amener la fin de la négociation :

- Si cet événement est l'interprétation d'un message reçu (le message de son partenaire), l'entité peut envoyer soit une *négociation*, soit une *conclusion* à son partenaire. Si le message est une *négociation*, cela signifie que l'entité veut encore re-négocier ; sinon la fin de négociation est conclue avec le message de type *conclusion*.
- Si cet événement implique une réaction, l'entité peut envoyer soit une *proposition*, soit une *conclusion* à son partenaire. Si le message est une *proposition*, cela signifie que son partenaire peut encore re-négocier ; sinon il met fin à la négociation avec un message de type *conclusion*.

6.3.4 Transformation des messages-événements

Le *protocole de négociation* est lié au module de *stratégie de négociation* par l' *Event Analyzer*. Le protocole travaille avec des messages, en revanche la *stratégie de négociation* traite des événements. Il faut donc transformer les événements en messages et les messages en événement. Cette fonctionnalité est prise en charge par le module *event analyzer*.

Si l'on s'intéresse à nouveau à l'exemple d'une transaction concernant un achat en ligne, nous voyons que lorsque la *stratégie de négociation* sélectionne l'événement *paiement*, cela se traduit par le fait que le *client* effectue le paiement au *marchand*. L'*Event Analyzer* envoie au marchand un message contenant toutes les informations nécessaires pour valider le paiement (numéro de carte bancaire et montant payé). Nous venons de voir que le mécanisme de négociation permet de re-négocier une action en envoyant un message de type *négociation* à son partenaire. En conséquence il est nécessaire d'en différencier les contenus.

Un événement peut être transformé en messages de type différents, selon son utilisation dans le processus de négociation. Considérons l'exemple suivant (ref : la structure d'événement du client dans la section 6.5) : lorsque l'événement *pay_ direct* est activé pour la première fois, l'*Event Analyzer* l'interprète comme une proposition de paiement, et un message de type *proposition* contenant l'ensemble des informations nécessaires est construit et envoyé. Par contre, lorsque cet événement apparait une deuxième fois dans le processus, l'*Event Analyzer*, sachant que l'on est dans une phase de re-négociation, l'interprète comme un message de type *négociation* qui ne contient que les informations liées au comportement effectué dans l'étape précédente (qui rappellent les caractéristiques du paiement précédemment effectué, par exemple en indiquant l'identifiant de la transaction bancaire).

6.4 Module de négociation de la confiance

Dans cette section, nous présentons le modèle de notre mécanisme de négociation. Il se compose de deux parties distinctes : la gestion de la confiance et la gestion de la négociation. La gestion de la confiance ayant été présentée en détail dans le chapitre 5, nous expliquons comment nous modélisons les activités de chaque entité à l'aide d'une structure d'événements, comment nous définissons leur politique et nous proposons une stratégie pour mener à bien les négociations.

6.4.1 Structure d'événements

Comme nous l'avons présenté au chapitre 5, nous utilisons une structure d'événements et une politique pour modéliser le comportement des différentes entités du système.

Aucun des participants ne connait la politique de l'autre ; une politique de confiance est quelque chose de personnel, on ne la partage pas, pour éviter que son correspondant n'en tire un profit concurrentiel par exemple (si mon correspondant connait le montant maximum que je suis prêt à dépenser pour acquérir un bien, c'est cette somme qu'il me réclamera même si elle est supérieure au montant qu'il comptait me demander).

Chaque entité doit définir sa propre structure d'événements. Un soin particulier doit être pris lors de la définition des relations de conflit et de causalité. Ces relations sont à la base du comportement de l'entité et doivent lui permettre des interactions riches avec ses correspondants.

On distingue deux types d'événements dans la structure d'événements :

- les événements qui spécifient des comportements (ou des actions) ; par exemple la commande de marchandise ou le payement d'un bien ;
- les événements qui ne portent que de l'information concernant la transaction ou l'entité engagée dans la transaction ; par exemple les événements qui représentent le risque ou la satisfaction.

Ces événements sont importants pour définir la politique de chaque entité. Étant pris en compte dans l'historique des transactions, ils sont utilisés lors de la vérification de la politique à chaque étape de la transaction.

Le choix de la structure d'événements est très important parce que cela a un impact direct sur l'arbre des transitions, et donc la façon dont les négociations seront menées. Le nombre d'événements a aussi son importance ; s'ils sont trop nombreux la rédaction des politiques est rendue difficile, et dans le cas contraire, les politiques risquent de ne pas être suffisament riches. Nous fournissons une analyse de ces différents aspect dans les sections concernées.

Poids relatif à la causalité

Pour spécifier notre processus de négociation, nous avons besoin de définir la notion de poids relatif à la *causalité d'un événement*

Le poids $P(e)$ de la causalité d'un événement e est la plus longue chaîne de conséquences possibles avec une structure d'événements donnée si on déclenche e.

Par exemple, si le déclenchement de l'événement e n'a pas de conséquence possible, son poids $P(e)$ est égal à 0 ; par contre si $e \leq e_1$, et $e_1 \leq e_2$ nous avons $P(e) = 2$ ($\leq$ est la relation de causalité entre deux événements).

6.4.2 Objectif de négociation

Comme nous l'avons vu dans la section 6.2, chaque entité a son objectif personnel pour la négociation. C'est un but que l'entité veut atteindre en engageant la négociation avec l'autre entité. Du point de vue du concepteur du système dans notre contexte d'application, nous pouvons le reconnaître facilement : *c'est un fait ou une observation indiquant que l'entité dispose de ce qu'elle attend.* Par exemple, dans le scénario de la transaction de vente-achat en ligne, l'objectif du client est de recevoir le produit envoyé

par le vendeur selon sa commande, celui du vendeur est de recevoir un paiement valide du client.

En modélisant le système avec comme modèle une structure appropriée, l'objectif est marqué dans la structure du modèle. Dans l'implémentation du système, l'objectif est codé dans la structure du programme. La section suivante explique comment l'*objectif* est modélisé dans cette structure.

***Définition* : objectif**

Dans la structure d'événements proposée pour l'entité en modélisant le système, *l'objectif est un événement que l'entité veut pouvoir observer.*

La structure d'événements traduit tous les *faits ou observations* des comportements de l'entité sous la forme d'événements. L'objectif se trouve donc dans cet ensemble d'événements. Pour le trouver dans la structure d'événements, l'événement « objectif » est marqué. Selon la modélisation du système en structure d'événements, il existe un ou plusieurs objectifs dans la structure. Dans la conception, les objectifs de l'entité sont précisés en modélisant le système.

Nous pouvons indiquer les objectifs de chaque entité en définissant leurs structures d'événements. Considérons le scénario de la transaction de vente-achat en ligne : l'objectif du client est de recevoir le produit commandé et ce fait est concrétisé avec l'événement *object_ received* de la structure d'événements du client, donné dans la section 6.5 ; de même, l'objectif du vendeur est concrétisé par les événements *payment_ received* et *payment_ TTP_ received* dans sa structure d'événements.

***Définition* : chemin d'objectif**

Le *chemin d'objectif* est *une configuration* possible qui contient un *objectif.*

Il peut y avoir plusieurs *chemins d'objectif* à partir des conséquences possibles d'une action.

Si à une étape de la négociation on observe l'événement e_p, on peut déterminer que l'événement e_s (qui est une conséquence de e_p) permet de contruire un *chemin d'objectif* en utilisant la procédure, illustrée par l'algorithme 1

6.4.3 Politique de confiance

Une politique de confiance est contruite en utilisant les événements décrits dans la structure d'événements, les évenements *informatifs* produits par les différents modules spécialisés et l'historique des transactions. La politique permet de spécifier les conséquences que l'on considère comme risquées et que l'on souhaite éviter. L'utilisation de l'historique permet de tenir compte de sa propre expérience et de modifier son comportement en conséquence.

Supposons que nous voulions que la séquence d'événements ea_i et eb_j (avec $ea_i \leq eb_j$) ne survienne jamais si nous observons l'*événement* e_r, il est possible de le spécifier par la politique suivante :

$$\psi \equiv G^{-1}[e_r \Longrightarrow \neg(ea_i \vee eb_j)]$$

La politique peut se réduire à l'expression de contraintes entre les événements dans la structure d'événements. La négociation peut être interprétée comme le processus de construction itératif d'une session, qui pour être correcte doit respecter les contraintes

```
Procédure CheminObjectif(e_s, x_n)
    // ES est la structure d'événements
    // l'ensemble E_c des e_ss qui peuvent être ajoutés en x_n
    E_C: ensemble d'événements
    si e_s est un objectif alors
        retourner true
    sinon
        E_C = {e_ss ∈ ES où (e_s ≤ e_ss ou e_ss¬#e_s) tel que (e_ss n'est pas en conflit
        avec x_n) ∧ (H ∪ e_ss ⊨ ψ)}
        si E_C est vide alors
            retourner false
        sinon
            pour e_ss ∈ E_C faire
                si CheminObjectif(e_ss, x_n) alors
                    e_ss ← x_n
                    retourner CheminObjectif(e_ss, x_n)
                fin
            fin
        fin
    fin
end
```

Algorithme 1: Chemin d'objectif

imposées par la structure d'événements. Cette dernière permet de contrôler le comportement des parties pour la session courante. La sous-politique est dans ce cas une formule sans quantificateur temporel et la vérification est réalisée uniquement en tenant compte de la session courante de l'historique.

Considérons l'exemple suivant : à un instant donné, l'entité A observe l'événement eb_i et elle veut que, si elle réagit en utilisant l'événement ea_i, cela impose le déclenchement de l'action correspondant à ea_{i+1}. Il est possible de spécifier cette contrainte avec la politique suivante :

$$\psi \equiv (eb_i \wedge ea_i) \Longrightarrow ea_{i+1}$$

Si l'on a défini un ensemble de sous-politiques différentes $\psi_1, \ldots, \psi_n$, qui prennent en compte différentes conséquences considérées comme risquées, la politque globale est une disjonction de cet ensemble de sous-politiques :

$$\psi_{global} \equiv \psi_1 \wedge \ldots \wedge \psi_n$$

6.4.4 Étape de négociation

La négociation est un processus d'échange. Chaque étape va consister à réagir à une action ou un ensemble d'actions successives de son partenaire. Lorsque l'on a observé un événement ou un ensemble d'événements successifs (qui représente le comportement de son partenaire), on recherche quels sont les événements appropriés de notre comportement qui permettent de poursuivre la négociation. Dans cette section, nous spécifions

d'abord ce qu'est une étape de la négociation, puis nous proposons un pseudo-code pour cette opération.

6.4.4.1 Spécification

Supposons que l'entité A soit à une étape de négociation où elle vient d'observer l'événement eb'_i pour la session x_n avec son partenaire B, étape illustrée par la figure 6.5. A peut réagir à cette action avec les événements ea_j ou ea_{j+1} qui respectent sa politique. Cela veut dire que :

$$H_A.\{\dots eb'_i, ea_j \dots\} \models \psi_A \text{ et } H_A.\{\dots eb'_i, ea_{j+1} \dots\} \models \psi_A$$

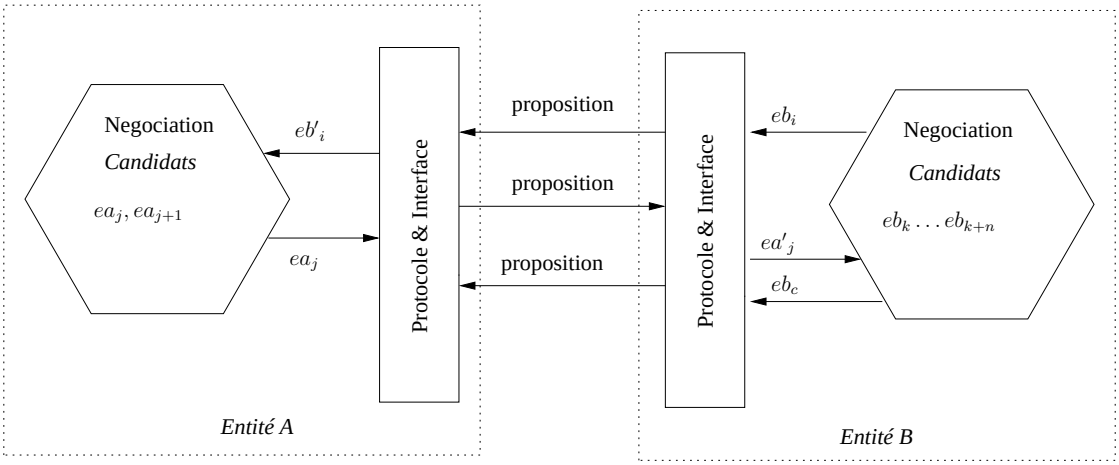


FIG. 6.5 – Étape de négociation

Supposons que à cette étape, la *stratégie de négociation* de A conseille l'utilisation de ea_j, il est ajouté à la session x_n de l'historique :

$$H_A = x_1 \dots x_{n-1}.\{\dots eb'_i.ea_j\}$$

Quand B observe l'action ea'_j (interprétation du message qu'il vient de recevoir), il peut réagir en envoyant à A un message construit à partir d'un des événements sélectionnés comme candidats potentiels $E_C = (eb_k, eb_{k+1} \dots eb_{k+n})$. Supposons que le gestionnaire de stratégie de B lui propose d'utiliser eb_c ($eb_c \in E_C$), mais que :

- eb_c ne lui permette pas d'atteindre son objectif ;
- eb_c soit un événement qui corresponde à une action qui termine la négociation avant que le but de la négociation ne soit atteint.

Dans ce cas, B veut *re-négocier* avec A pour qu'il réagisse d'une manière différente. B va se comporter de la manière suivante :

- B ne prend pas en compte ea'_j dans son historique,
- B recherche dans la session courante le dernier événement qu'il a pris en compte, noté eb_i, qui correspond à la réponse qu'il a faite à A lors de l'étape précédente.
- B envoie de nouveau à A un message correspondant à l'événement eb_i.

A observe de nouveau l'événement eb'_i (interprétation que A fait du message envoyé par B et correspondant à eb_i). Il détecte qu'il y a un problème, l'événement eb'_i étant

déjà présent dans la session x_n. A comprend que la réaction qu'il a eue à cette action n'est pas compatible avec la politique ou les objectifs de B (c'est la manière dont il faut interpréter la situation). A doit donc ré-évaluer la situation. Si A ne dispose pas d'un nouveau candidat, c'est qu'il avait réagi de la seule manière qui lui était possible à l'étape précédente ; il est alors obligé de mettre fin à la transaction. Sinon, A réalise les opérations suivantes :

- A supprime de la session x_n l'événement qui vient de déclencher cette réaction ; il s'agit de ea_j dans ce cas.
- A choisit un autre événement dans l'ensemble des candidats : ea_{j+1} par exemple.
- A reprend la négociation en envoyant à B un message correspondant à l'événement ea_{j+1}.

6.4.4.2 Pseudo-code d'une étape de négociation

Le pseudo-code suivant présente la procédure de négociation que nous venons de décrire lorsque l'on observe le comportement présenté dans l'algorithme 2

6.4.4.3 Négociation et messages envoyés plusieurs fois

Nous avons vu que que lors de l'étape de re-négociation, un événement est re-proposé au partenaire pour solliciter une autre réponse de sa part. Comme nous l'avons présenté dans la section 6.3, lors d'une re-négociation, le message porte le type *négociation*, son contenu est donc facilement différenciable d'un message de type *proposition* par l'*Event Analyzer*.

6.4.5 Procédure d'envoi et d'observation des événements

Pour simplifier la spécification des échanges, nous proposons deux procédures qui permettent de manipuler les événements : envoyer ou observer des événements. Ces procédures sont partie intégrante de l'*Event-Analyzer* et sont chargées d'assurer l'interface entre messages et événements.

Envoyer(e) : cette procédure permet d'envoyer à son partenaire un message m_e correspondant à un événement e. Lors de l'appel de la procédure, l'*Event Analyzer* interprète l'événement e et le met sous la forme d'un message m_e ; le message est envoyé à l'autre partie.

e=Observer() : cette fonction rend un événement qui correspond à un message observé lors du processus d'échange. Lorsque l'entité reçoit le message m_e, l'*Event Analyzer* l'interprète sous la forme d'un événement e, cet événement est fourni au module de négociation pour les traitements nécessaires par la suite.

6.4.6 Processus d'échange

Nous présentons le processus d'échange complet lorsque deux parties utilisent leur structure d'événements pour la négociation. Les données échangées sont des *messages* comme nous venons de l'expliquer dans la section *Protocole de négociation*.

Supposons que deux entités A et B veuillent réaliser une transaction. A est à l'initiative de la négociation. Elles disposent chacune de leurs historiques respectifs H_A et H_B et de leurs politiques ψ_A et ψ_B. Le processus d'échange est réalisé en plusieurs étapes et le processus de négociation est le suivant :

Procédure NégociationEtape($(\ldots eb_{i-1}, eb_i)$, ψ_A)
  E_C: **événements candidat**
  x_n: **session courante de l'historique**
  cas où $(\ldots eb_{i-1}eb_i)$ *n'est pas conflit avec* x_n *: // première proposition*
    pour $e_s \in ES$ *tels que* ($eb_i \leq e_s$ *ou* e_s *indépendant* eb_i) $\wedge$(e_s *n'est pas conflit* x_n) **faire**
      $x_n \leftarrow e_s$ // temporaire
      si $H_A \models \psi_A$ **alors**
        $E_C \leftarrow e_s$
      fin
    fin
    si $|E_C| > 1$ **alors**
      // *L'étape que A négocie avec B pour avoir une autre conséquence possible*
      // *Appliquer la procédure* $cheminObjectif(e_s)$ *pour vérifier si* e_s *mène à un événement d'objectif*
      si $\exists e_s \in E_C$ *tel que* $cheminObjectif(e_s) = vrai$ **alors**
        ea_j = StratégieNégociation$^{**}(eb_i, E_C)$ //appliquer la stratégie
        $x_n \leftarrow ea_j$
        $E_C \rightarrow ea_j$ //Supprimer ea_j de E_C
        Envoyer$^{**}(ea_j)$
      sinon
        Chercher le dernier $ea_{j-1} \in x_n$ // dernier envoi à B
        Envoyer(ea_{j-1}) // envoyer à nouveau ea_{j-1} pour négocier
      fin
    fin
  fin
  cas où $eb_i \in x_n$ *: // le partenaire demande une re-négociation*
    si $|E_C| = 0$ **alors**
      // Un seul événement candidat qui a déjà été proposé
      $FIN \leftarrow true$
    sinon
      $x_n \rightarrow ea_j$ // Supprimer ea_j de x_n ;
      ea_{j+1} = StratégieNégociation$^{**}(eb_i, E_C)$
      $x_n \leftarrow ea_{j+1}$
      $E_C \rightarrow ea_{j+1}$ //Supprimer ea_{j+1} de E_C
      Envoyer(ea_{j+1})
    fin
  fin
end
** : *ces fonctions sont décrites dans les sections suivantes*

Algorithme 2: Procédure d'une étape de négociation.

Spécification des étapes

1. Avant de commencer le processus de négociation, A et B ajoutent une nouvelle session à leur historique respectif : $H_A = H_A \cup x_a$ et $H_B = H_B \cup x_b$.
2. A débute l'échange par une *proposition*, l'événement correspondant est ea_1. Cet événement est ajouté à x_a : $x_a = x_a.ea_1$. Un message est envoyé à B pour l'informer (appel de la procédure : $Envoyer(ea_1)$).
3. B reçoit le message envoyé par A et l'interrogation de la fonction $Observer()$ rend à cette étape l'événement ea'_1. Cet événement est ajouté à x_b. B utilise sa procédure de négociation pour déterminer quelle réaction avoir ($eb_1 = NegotiationEtape(ea'_1, \psi_B)$). Si cet événement indique la fin de la transaction, le protocole d'échange se termine pour B sinon, B informe A de sa décision (appel la procédure $Envoyer(eb_1)$.
4. À l'étape i, A observe un nouvel événement : $eb'_i = Observer()$; il utilise sa procédure $NegotiacionEtape(eb'_i, \psi_A)$ pour déterminer la réaction appropriée. Ce processus continue jusqu'a ce que les deux parties atteignent la fin du processus de négociation.

En appliquant cet algorithme, nous trouverons que la fin de négociation est atteinte lorsque la session courante est une configuration maximale. Nous pouvons vérifier que l'objectif de la négociation est atteint en vérifiant la présence d'un événement d'objectif dans la session.

Pseudo-code

Voici le processus complet de négociation de l'entité A en négociation avec son partenaire B.

```
Procédure ProcessNégociation(ψ_A, H_A, debut : booleen)
    // debut : précise si A débute la négociation
    E_C: événements candidats
    x_a: session créée pour cette transaction
    H_A = H_A ∪ x_a
    si debut = true alors
        x_a ← ea_1 // débuter
        Envoyer(ea_1) //événement débutant la négociation
    fin
    tant que FIN = False faire
        eb'_i=Observe()
        NegotiacionEtape(eb'_i, ψ_A)
    fin
end
```

Algorithme 3: Procédure de négociation

6.4.7 Stratégie de négociation

Cette section explique comment une entité peut mettre en œuvre sa stratégie à chaque étape de négociation. La stratégie de négociation est une méthode qui doit permettre de sélectionner le meilleur événement parmi les événements candidats à l'étape

suivante. Afin de rendre palpable cette notion de stratégie, nous proposons au lecteur un exemple d'algorithme.

Nous avons vu que l'ensemble des événements candidats E_C est composé d'événements e_{ss} qui satisfont les contraintes suivantes :

- ils respectent les relations de causalité et de conflit avec les événements de x_n (x_n est la session courante) ;
- la politique continue à être satisfaite.

L'acteur peut décider de programmer la stratégie suivante dans son module *Strategy Negotiation*. Nous distinguons deux cas : l'acteur a atteint son objectif ou ne l'a pas encore atteint (un événement *objectif* est présent ou non dans la session courante).

1. L'acteur n'a pas encore atteint son objectif :
   - Sauf si c'est impossible, l'acteur évite de sélectionner un événement qui l'amène à mettre fin à la négociation.
   - L'entité veut atteindre son objectif le plus tôt possible. S'il existe une relation causale (directe ou indirecte) entre un événement e_j et un événement d'objectif e_{ob}, elle choisit l'événement dont la longueur du *chemin d'objectif* est le plus court.
   - L'entité peut également choisir le e_j dont le poids relatif à la causalité est le plus faible. En choisissant cet événement, l'entité a une chance d'atteindre plus rapidement une configuration maximale
2. L'acteur a atteint son objectif et son partenaire ne l'a pas encore atteint. Si l'acteur est honnête, il doit éviter de sélectionner un événement qui mène à la fin de la négociation s'il existe d'autres possibilités. Terminer la négociation empêche toute re-négociation de la part de son partenaire. Par contre, il peut choisir un e_j de poids minimal pour terminer au plus tôt la négociation. Dans ce cas, il envoie une *proposition* à son partenaire pour lui permettre d'être à l'initiative de la fin de la transaction.

Nous pouvons écrire cet algorithme dans le cas où l'entité veut terminer au plus tôt la négociation :

```
Fonction StratégieNégociation(E_C, e_p): événement
    // E_C : ensemble non vide des événements candidats après avoir observé e_p
    pour e_cj ∈ E_C faire
        // Calculer le poids de causalité des candidats
        P(e_cj) ← poids-causalite (e_cj)
    fin
    si |E_C| > 1 alors
        // l'événement e donne la FIN si il complète la configuration maximale
        Choisir e ∈ E_C tel que (e ne donne pas la FIN) et P(e) plus petit
        retourner e
    sinon
        retourner e unique de E_C
    fin
end
```

Algorithme 4: Procédure de la stratégie

Hasard

La stratégie qui consisterait à choisir au *"hasard"* l'événement de l'ensemble E_c à utiliser pour l'étape suivante de la négociation est aussi tout à fait utilisable. Cette stratégie a des propriétés intéressantes, comme celle de ne privilégier aucune direction de négociation particulière, mais va aussi rendre difficilement prévisible le comportement d'un pair, ce qui peut être considéré comme un atout dans certaines situations sensibles (si le fait de pouvoir prédire le comportement de son correspondant vous offre un avantage stratégique ou concurrentiel).

6.4.8 Discussion sur le modèle

Les messages échangés par les deux acteurs peuvent contenir des données sensibles qui doivent être protégées. Nous ne traiterons pas ce problème dans le cadre de cette thèse, les protocoles de sécurité traditionnels utilisant des schémas cryptographiques pouvant être utilisés à cette fin. Une solution pratique, simple et éprouvée que l'on peut mettre en œuvre est de chiffrer l'ensemble des communications en utilisant un protocole construit au dessus de SSL. Nous n'avons besoin que d'assurer la confidentialité des échanges, les aspects liés à l'authentification des correspondants n'ayant pas d'objet dans notre cas : nous sommes en train de tenter d'établir une relation de confiance avec quelqu'un que nous ne connaissons pas.

Nous avons proposé à titre d'exemples des stratégies de négociation qui peuvent s'appliquer dans un contexte général. Dans le cadre d'applications spécifiques, il est toujours possible de proposer d'autres stratégies qui correspondraient mieux aux objectifs des participants, et qui permettraient d'avoir des négociations plus efficaces.

Dans cette modélisation, nous avons considéré la phase de re-négociation de l'entité pour une seule étape de retour en arrière. Il est possible de modéliser cette re-négociation pour pouvoir revenir sur plusieurs étapes précédentes. Cela dépend de la politique proposée par chaque partie.

6.5 Scénario d'application

Dans cette section, nous allons illustrer le fonctionnement de notre mécanisme de négociation avec un exemple de transaction dans le domaine du commerce électronique. Nous détaillons d'abord le scénario puis nous en présentons le déroulement selon les principes de notre système de négociation.

6.5.1 Transaction de e-commerce en ligne

Nous imaginons un système de commerce en ligne qui permette à des vendeurs et des clients de mener à bien des transactions. Nous allons étudier le cas des transactions où un client veut acheter un produit à un marchand. Les deux parties veulent pouvoir réaliser la transaction en toute confiance : le client veut être sûr qu'il recevra bien le produit s'il l'a payé et le vendeur veut être sûr d'être payé s'il livre le produit au client.

L'acheteur et le vendeur ont besoin de négocier pour réaliser une transaction de ce type. La négociation se déroule sous la forme d'une suite d'échanges de messages entre les deux parties tout en respectant la politique de confiance de chacun. Dans le cas général, nous pourrions observer la séquence d'échanges suivante :

– L'acheteur envoie une commande pour un produit au vendeur pour débuter la transaction.

– Le vendeur analyse la commande (disponibilité du produit, profil de l'acheteur ...). Si les conditions de la commande sont valides, le vendeur envoie une demande de paiement à l'acheteur.
– L'acheteur analyse à son tour cette réponse pour décider s'il continue la transaction. Il peut vérifier le profil du vendeur, évaluer le niveau de risque de transaction ... Il peut décider de payer le vendeur, lui demander des informations complémentaires ou interrompre la transaction.
  Le client peut utiliser différentes méthodes pour payer le vendeur : le mode de paiement que l'on appelle *paiement direct* où le montant du paiement est directement crédité sur le compte du vendeur (virement bancaire ou carte VISA par exemple) ; le mode de paiement utilisant les services d'un tiers de confiance. Dans cette situation, le paiement est versé au tiers de confiance qui conserve la somme tant que la transaction n'est pas confirmée. Nous modélisons un tiers de paiement de confiance dans la section 6.5.2.
– À la suite de l'étape précédente, si le paiement est réalisé avec le mode de paiement *direct*, le vendeur reçoit directement le paiement. Mais si le paiement est réalisé par l'intermédiaire d'un tiers de confiance, lorsqu'il reçoit la marchandise, le client envoie l'ordre de paiement au tiers de confiance qui reverse alors le montant de la transaction au vendeur.
– À la fin de la transaction, chaque entité peut évaluer son partenaire de manière à enrichir sa propre expérience.

Le schéma de transaction est illustré à la Figure 6.6.

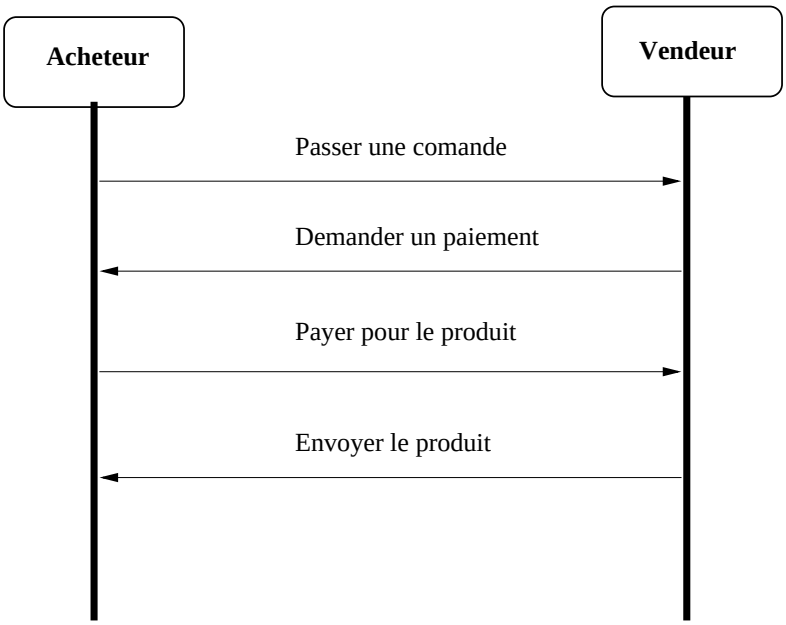


FIG. 6.6 – Scénario : transaction vente-achat en ligne

Pour simplifier le scénario, nous ne considérons que les transactions dont le déroulement respecte correctement la description ci-dessus. Bien qu'elles soient possibles, nous excluons les transactions irrégulières suivantes :

– le client reçoit un produit qu'il n'a pas commandé ;

– le client reçoit une demande de paiement pour un produit qu'il n'a pas commandé ;
– le vendeur reçoit le paiement de quelque chose qui n'a pas été commandé ;
– lee vendeur n'envoie pas le produit à un client qui l'a commandé.

Nous pouvons bien sûr exclure les transactions de ce type en proposant des politiques qui l'interdisent. Mais le but de l'étude de ce scénario est de présenter le fonctionnement d'une négociation entre un client et un vendeur, la complexification des politiques aurait allourdi inutilement l'étude de cas.

Pour une transaction, l'objectif du client est de recevoir le produit commandé et l'objectif du vendeur est d'obtenir le paiement de ce produit.

6.5.2 Tiers de confiance pour le paiement

Comme nous l'avons décrit dans la section précédente, lorsque le client paie le vendeur pour une transaction, il peut utiliser le mode de *paiement par l'intermédiaire d'un tiers de confiance* (trusted third party). Un tiers sûr de paiement est un service qui permet au client de contrôler son paiement en tenant compte des événements qui peuvent survenir pendant la transaction.

Le service fonctionne comme une assurance : celle d'être payé pour le marchand et celle d'être remboursé en cas de problème pour le client. On peut tout à fait admettre que l'accès à ce service ne soit pas gratuit et que des frais puissent être prélevés par le tiers de confiance lors de chaque transaction. En cas de transaction menée correctement à son terme, le marchand paye un pourcentage du montant au tiers de confiance et en cas de problème, le client paye lui aussi un pourcentage du montant de la transaction pour récupérer ses fonds. Cette somme prélevée par le tiers de confiance correspond au coût de l'assurance.

Cas d'un paiement correct

Les étapes suivantes sont réalisées par le tiers de confiance lorsque la transaction s'établit normalement du point de vue du client :

– Le client dépose le montant du paiement chez le tiers, en indiquant à quel marchand et pour quelle transaction cette somme est déposée. Le client et le vendeur sont tous les deux des utilisateurs de ce service ; ils y disposent d'un compte les identifiant.
– Lorsque le client dépose la somme pour le paiement, le tiers de confiance lui fournit un *code* qu'il pourra transmettre au vendeur par la suite. Le vendeur le présentera au tiers de confiance pour pouvoir récupérer le montant de l'achat. À ce stade, le tiers notifie également au vendeur que le client lui a bien transféré les fonds nécessaires à la transaction.
– Le client envoie le code de paiement au vendeur comme confirmation du paiement.
– Le vendeur présente ce code de paiement au tiers de confiance, et à partir de ce moment le tiers peut verser au marchand la somme correspondant au paiement liée à la transaction.

Cas d'un paiement incorrect

Le tiers propose un mécanisme pour régler le paiement du client au vendeur en cas de problème défavorable arrivé pour une transaction, pour le cas où soit le client, soit le vendeur est malhonnête.

– Si le vendeur est malhonnête, après avoir reçu la notification de paiement du tiers de confiance, il n'envoie pas le produit au client. Le client veut alors récupérer la somme d'argent déposée en ayant peut-être à payer une commission.
– Si le client est malhonnête, il ne transmet jamais le code de confirmation de paiement au vendeur. Dans ce cas, le vendeur peut réclamer le paiement au tiers de confiance en lui présentant les preuves attestant de la transaction (bon de livraison signé par exemple). Une commission sur le montant de la transaction est prélevée par le tiers de confiance.

Il existe plusieurs services sûrs de tiers de paiement en pratique : le système de PayPal [45], le système de transfert de WesternUnion [4]... Chaque système peut proposer sa propre politique pour sa rémunération, le paiement et le traitement des réclamations des clients ou des vendeurs.

6.5.3 Observations et structures d'événements

Nous proposons dans cette section un modèle d'observation de la transaction. Celui-ci est décrit par la structure d'événements définie par chacune des entités, le *vendeur* et l'*acheteur*. Nous précisons que le modèle d'observation de chaque entité lui est propre et dépend de la structure d'événements qu'il a choisie. Nous présentons la structure d'événements de l'acheteur dans la figure 6.7 et celle du vendeur figure 6.8.

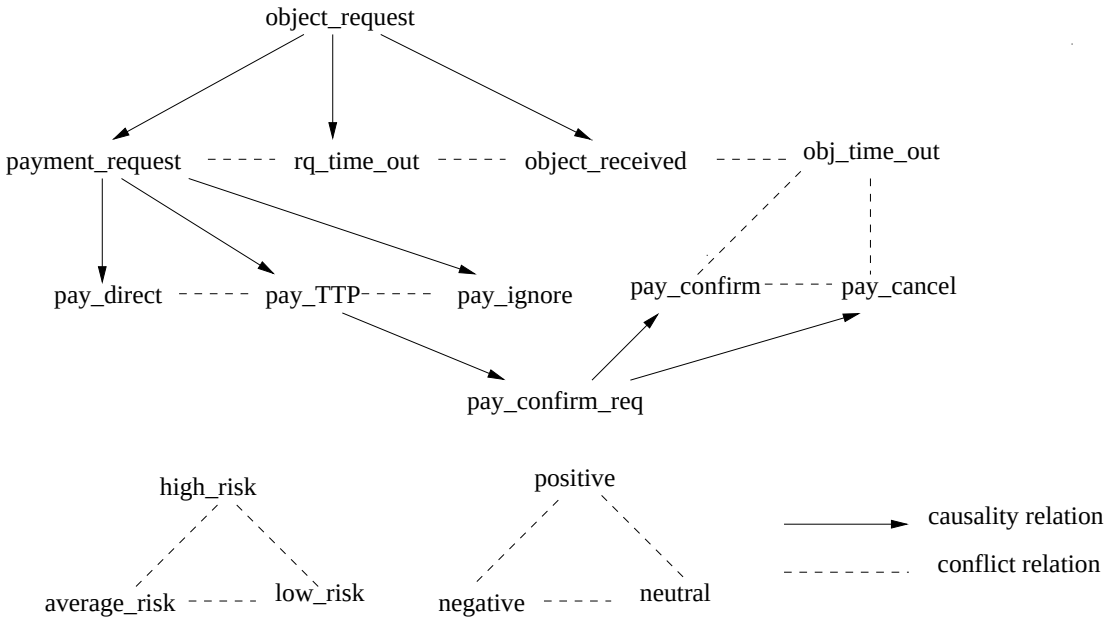


FIG. 6.7 – Acheteur : Structure d'événements

La structure d'événements de chaque entité décrit les relations de *conflit* et de *causalité* que doivent respecter les événements. Les points suivants présentent des explications détaillées sur chacun des événements de la structure d'événements de l'acheteur et du vendeur.

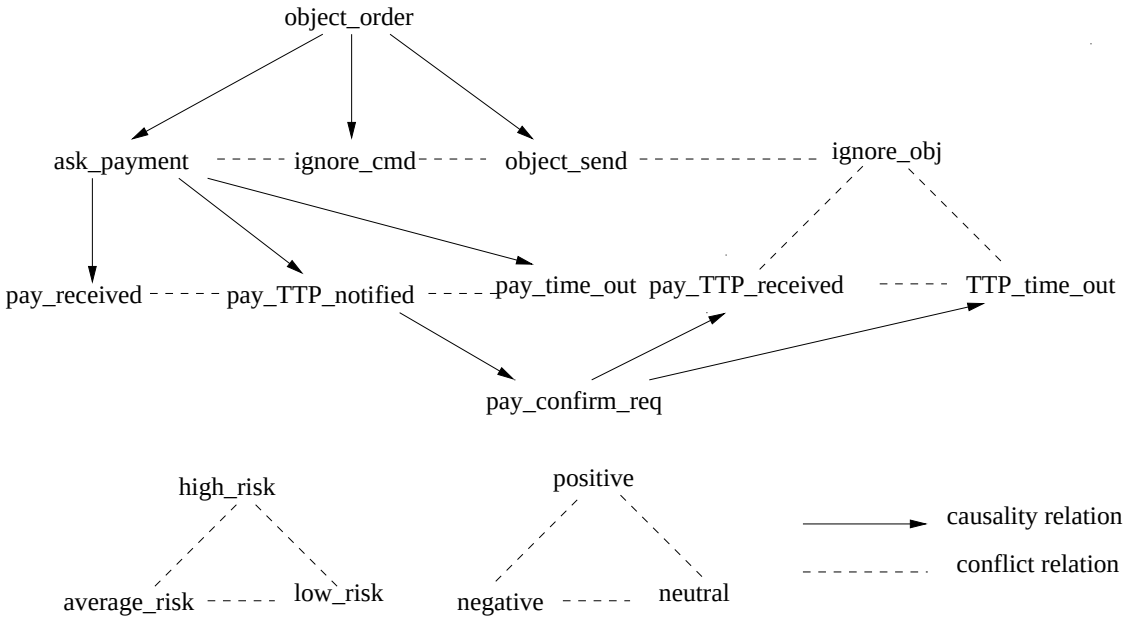


FIG. 6.8 – Vendeur : Structure d'événements

Acheteur : structure d'événements

La spécification des événements de la structure d'événements de l'acheteur est donnée ci-dessous.

- *object_ request* : événement local indiquant que l'acheteur passe une commande à un vendeur.
- *object_ received* : indique qu'il reçoit un message contenant une confirmation du paiement et un plan d'envoi du produit de la part du vendeur.
- *obj_ time_ out* : indique qu'il n'a reçu aucune information du vendeur après avoir passé sa commande.

- *high_ risk, average_ risk, low_ risk* : événements locaux qui décrivent le niveau de risque de la transaction (élevé, moyen ou faible). Ces événements sont déterminés par le *risk manager* qui sera présenté dans le chapitre 7. Ce composant utilise les informations disponibles sur la transaction (coût total, probabilité d'échec...) pour évaluer le niveau de risque.

- *positive, neutral, negative* : évaluation de la qualité de la transaction par le client.

- *payment_ request* : l'événement qui indique que le vendeur a envoyé une demande de paiement suite à la commande.
- *pay_ direct* : événement qui indique que le paiement est effectué directement (carte bancaire, argent liquide...). Cet événement fait suite à la demande de paiement du vendeur.
- *pay_ TTP* : événement qui indique que le paiement est effectué en ayant recours aux services d'un tiers de confiance.
- *pay_ ignore* : indique qu'il ignore la demande de paiement du vendeur.
- *pay_ confirm* : indique qu'il confirme le paiement auprès du tiers de confiance.

- *pay_ confirm_ req* : l'événement qui indique que le vendeur demande une confirmation de paiement
- *pay_ cancel* : indique que le vendeur ne valide pas son paiement auprès du tiers de confiance.

À partir de cet ensemble d'événements et l'objectif du client, nous pouvons trouver que l'événement qui spécifie l'objectif du client dans la structure d'événements est *object_ received*.

Vendeur : structure d'événements

Nous trouvons ci-dessous la spécification de la structure d'événements du vendeur.

- *object_ order* : indique que le vendeur reçoit une commande d'un acheteur.
- *object_ send* : événement qui indique que le vendeur a envoyé le produit à l'acheteur.
- *order_ ignore* : événement local qui indique que le vendeur ignore la commande du client. Cela peut survenir dans le cas d'une commande invalide ou dans le cas d'un client considéré comme étant un escroc, un voleur...

- *ask_ payment* : l'événement indiquant que le vendeur réclame le paiement de la commande à l'acheteur.
- *payment_ received* : indique que le vendeur a reçu le paiement de l'acheteur par la méthode de paiement direct.
- *payment_ TTP_ received* : indique que le vendeur a reçu le paiement avec la méthode de paiement par du tiers de paiement.
- *pay_ TTP_ notified* : indique que qu le vendeur reçoit une notification de paiement par un tiers de confiance.
- *payment_ time_ out* : indique qu'il n'a toujours pas reçu de paiement après un délai fixé.

- Les autres événements ont la même sémantique que ceux décrits pour la structure d'événements de l'acheteur.
- *ask_ pay_ confirm* : l'événement indiquant que le vendeur demande une confirmation de paiement l'acheteur.

À partir de cet ensemble d'événements et de l'objectif du vendeur, nous voyons que les événements d'objectif du vendeur dans la structure d'événements sont *payment_ received* et *payement_ TTP_ received*.

6.5.4 Politique de confiance

Grâce à sa structure d'événements, chaque entité qui participe à la transaction (vendeur et acheteur) peut construire sa propre politique de confiance. Cette politique lui permet de vérifier que, à chaque étape, la négociation se déroule en satisfaisant correctement sa politique. La politique est exprimée sous la forme d'une formule logique PP-LTL qui décrit le comportement attendu tout au long de la transaction.

Politique de l'acheteur

La politique que l'acheteur met en place tend à minimiser les risques qu'il prend. Il refuse de prendre des risques avec les gens qu'il ne connaît pas, si les pertes financières possibles sont trop conséquentes ou s'il a déjà eu des problèmes avec le vendeur.

Avec la structure d'événements du client présentée par la figure 6.7, nous constatons que le client peut éviter deux conséquences risquées qui peuvent survenir s'il effectue le paiement après la demande du vendeur :

- la conséquence $pay\_direct \rightarrow obj\_time\_out$ indique que le client a payé par un règlement direct mais que le vendeur n'a pas envoyé le produit ;
- la conséquence $pay\_TTP \rightarrow obj\_time\_out$ indique que le client a payé par l'intermédiaire d'un service de confiance mais que le vendeur n'a pas envoyé le produit.

Pour éviter cela, en tenant compte des informations liées aux risques et à l'historique des transactions, le client peut proposer les politiques suivantes pour éviter de voir apparaitre ces conséquences lors de la négociation.

- Lorsqu'il reçoit une demande de paiement du vendeur, le client n'effectue pas de paiement direct s'il trouve que dans ses transactions antérieures avec ce vendeur il y a déjà eu $pay\_direct \rightarrow obj\_time\_out$, ou si le risque lié à la transaction est elevé.
  $\psi_1 : pay\_direct \Longrightarrow \neg(payment\_request \wedge$
  $high\_risk \wedge F^{-1}(pay\_direct \wedge obj\_time\_out))$
- Lorsque le risque est bas, ou lorsque le risque est moyen et que dans le passé il n'y a jamais eu $pay\_direct \rightarrow obj\_time\_out$, le client n'utilise pas les services du tiers de confiance.
  $\psi_2 : pay\_TTP \Longrightarrow payment\_request \wedge [low\_risk$
  $\vee(average\_risk \wedge G^{-1}(pay\_direct \wedge obj\_time\_out)]$

La politique globale de l'acheteur serait alors : $\psi_A \equiv \psi_1 \wedge \psi_2$.

Politique du vendeur

De la même manière, le vendeur veut se protéger des acheteurs indélicats et minimiser ses risques de pertes financières. Nous utilisons la structure d'événements pour spécifier d'abord les conséquences à risque pour le vendeur.

Dans l'arbre des transitions, nous voyons que le vendeur a intérêt à éviter les conséquences qui contiennent les chaînes suivantes :

- $object\_send \rightarrow TTP\_time\_out$: le client a payé par un tiers de confiance, le vendeur a envoyé le produit mais il n'y a jamais eu de confirmation du paiement.
- $object\_send \rightarrow pay\_time\_out$: le vendeur a envoyé le produit avant de demander un paiement au client et le client ne le paie pas.
- $ask\_pay\_confirm \rightarrow TTP\_time\_out$: le vendeur a envoyé le produit avant d'avoir le paiement ; le client a payé par le tiers de confiance mais ne confirme jamais le paiement.
- $obj\_order \rightarrow ignore\_cmd$: cas où le vendeur reçoit des commandes de clients considérés comme *indélicats*.

En utilisant les informations de risque et l'historique, le vendeur peut proposer les politiques suivantes pour éviter ces situations :

- Le vendeur n'accepte pas les commandes de la part de clients qu'il a évalués de manière négative dans le passé. $\psi_1 : G^{-1}(negative) \Longrightarrow obj\_order \wedge ignore\_cmd$
- Si le vendeur accepte d'envoyer le produit au client avant d'en avoir le paiement, il envoie en même temps la demande de paiement. $\psi_2 : (object\_order \wedge object\_send) \Longrightarrow ask\_payment$
- Le vendeur envoie le produit au client avant de demander un paiement seulement dans le cas où les risques sont faibles et s'il n'y a pas eu d'évaluation négative sur une transaction avec ce client. $\psi_3 : object\_send \Longrightarrow [low\_risk \wedge F^{-1}(negative)]$
- Le vendeur n'accepte que le mode de paiement par un tiers de confiance si les risques sont élevés et si dans le passé, il n'a pas eu de problème de paiement après l'envoi du produit.
  $\psi_4 : [high\_risk \wedge F^{-1}(object\_send \wedge pay\_cancel)]$
  $\Longrightarrow TTP\_notified$

La politique globale du vendeur est donc : $\psi \equiv \psi_1 \wedge \psi_2 \wedge \psi_3 \wedge \psi_4$.

6.5.5 Stratégie de négociation

Après avoir présenté la structure d'événements et la politique propre de chaque partie du protocole, nous allons détailler la phase de négociation et l'utilité du module de gestion de la Stratégie de Négociation. La négociation est décrite par la succession des événements sélectionnés à chaque étape du processus. Nous allons dérouler le processus de négociation pour deux situations : l'achat d'un morceau de musique en *MP3* et l'achat d'un *Smartphone*. La différence entre ces deux scénarios est liée au montant de la transaction et aux risques qui peuvent être perçus différemment par les participants. Nous rappelons que le processus de négociation utilise l'historique des interactions propre à chacun des participants et que les observations se traduisent par la mise à jour de la session courante de cet historique.

Supposons que l'historique du client et celui du marchand soient respectivement H_c et H_m au moment de la transaction :

$$\begin{aligned} H_c = &\{object\_request, average\_risk, payment\_request, object\_received\} \\ &\{object\_request, low\_risk, payment\_request, object\_time\_out\} \\ &\{object\_request, high\_risk, payment\_request, pay\_TTP, \\ &object\_received, pay\_confirm\} \end{aligned}$$

$$\begin{aligned} H_m = &\{object\_order, low\_risk, object\_send, pay\_time\_out\} \\ &\{object\_order, average\_risk, aks\_payement, payment\_received, object\_send\} \\ &\{object\_order, high\_risk, payment\_request, pay\_TTP, TTP\_notified, \\ &object\_send, payement\_received\} \end{aligned}$$

Il nous faut préciser que l'historique du client et celui du vendeur peuvent ne pas contenir le même nombre de sessions. Le client et le vendeur peuvent ne sauvegarder que les sessions qu'ils considèrent comme importantes.

Transaction pour une chanson MP3

Pour la transaction concernant l'achat d'une chanson MP3, l'acheteur et le vendeur peuvent considérer que le niveau de risque est bas. Cela nous conduit à une situation

où le *Risk Evaluator* de chaque participant fournit l'événement *low_risk* pour la transaction.

C'est le client qui débute la négociation. Le processus de négociation passe par les étapes suivantes (*client* et *marchand*) :

Étape 1, côté client

- Pour débuter la négociation, une nouvelle session, x_c est ajoutée à l'historique H_C. Le *client* envoie la commande, son action est prise en compte et ajoutée à x_c, $x_c = \{object\_request\}$.
- Le client évalue immédiatement le risque, il trouve *low_risk* pour ce cas et cela est aussi pris en compte en x_c, $x_c = \{object\_request, low\_risk\}$

Étape 2, côté marchand

- Pour prendre en compte la transaction, une nouvelle session x_m, est également ajoutée à l'historique H_m. Le *marchand* reçoit la commande, l'événement *object_order* est ajouté à x_m, $x_m = \{object\_order\}$
- Le *marchand* évalue immédiatement le risque, il trouve *low_risk* pour ce cas et c'est aussi pris en compte dans x_m, $x_m = \{object\_order, low\_risk\}$
- Ensuite, le *marchand* sélectionne un événement qui correspond avec sa politique pour réagir. Cette vérification fournit *ask_payment* comme le résultat, il est pris en compte. La session x_m actuelle est :
  $x_m = \{object\_order, low\_risk, ask\_payment\}$.

Étape 3, côté client

- Lorsque le *client* reçoit la demande de paiement, il met son historique à jour ($x_c = \{object\_request, low\_risk, pay\_request\}$).

- En confrontant sa politique de confiance à son historique, le *client* trouve que les actions correspondant aux événements *pay_direct*, *pay_TTP* et *pay_ignore* sont des candidats possibles.
- La *Stratégie de Négociation* sélectionne *pay_direct* pour réagir auprès du marchand ; l'événement est ajouté à l'historique et le message envoyé :
  $x_c = \{object\_request, low\_risk, pay\_request, pay\_direct\}$.

Étape 4, côté marchand

- Le marchand interprète le message du client sous la forme de l'événement *payment_received*. A ce stade, sa session courante est :
  $x_m = \{object\_order, low\_risk, ask\_payment, payment\_received\}$.
  Après vérification de sa politique, il trouve que l'événement *send_objet* est une réaction appropriée. Il envoie l'objet commandé au client et ajoute *send_objet* à x_m : $x_m = x_m.send\_objet$
- À ce stade, le marchand se trouve à la fin de la négociation.

Étape 5, côté client

- Le client reçoit l'information et interprète le message comme la réception de l'objet commandé sous la forme de l'événement *object_received* et le rajoute à son historique :
  $H_c = \{object\_request, low\_risk, pay\_request, pay\_direct, object\_received\}$.
- À ce stade, le client se trouve à la fin de la négociation.

Fin de la négociation

– Le client ne peut plus qu'évaluer la transaction en ajoutant à son historique un des événements *positive*, *neutral* ou *negative*. Son historique actuel :

$$H_c = \{object\_request, low\_risk, pay\_request, object\_received, positive\}$$

Le contenu de la session est maintenant une configuration maximale (il n'est plus possible de rajouter d'événements tout en respectant les relations de *conflits* et de *causalités*). Cela marque pour lui la fin de la transaction et de la négociation.

– De la même manière, le marchand n'a plus que la possibilité d'évaluer la transaction. Après l'ajout de son appréciation, son historique devient une configuration maximale qui marque pour lui aussi la fin de la transaction :

$$H_m = \{object\_order, low\_risk, ask\_payment, payment\_received, send\_object, positive\}$$

Transaction pour un *Smartphone*

Le prix d'un *Smartphone* est élevé par rapport à celui d'un morceau de musique, on imagine facilement que le client et le marchand puissent considérer qu'il s'agisse d'une transaction risquée. Nous sommes dans la situation où le module *Risk manager* du client lui rend un événement $high\_risk$ pour la transaction alors que celui du marchand rend $average\_risk$. Cela va donc avoir une influence sur le déroulement de la négociation.

C'est toujours le client qui débute la négociation. Le processus de négociation passe successivement par les étapes suivantes (*client* et *marchand*) :

Étape 1, côté client

– Une nouvelle sessions x_c est ajoutée à H_c. Le *client* envoie la commande et met à jour son historique : $x_c = \{object\_request\}$.
– En même temps, il évalue le risque et le prend en compte dans l'historique

Étape 2, côté marchand

– De la même manière, une nouvelle session x_m est ajoutée à H_m. Quand le *marchand* reçoit la commande, l'événement $object\_order$ est ajouté à l'historique : $x_m = \{object\_order\}$
– Le risque est évalué à cette étape comme $average\_risk$ et pris en compte dans x_m : $x_m = x_m.average\_risk$
– A la suite de la commande, le *marchand* ajoute l'événement $ask\_payment$ à son historique et envoie le message correspondant au client. La session courante est : $x_m = \{object\_order, average\_risk, ask\_payment\}$.

Étape 3, côté client

– Le *client* reçoit une demande de paiement et met à jour son historique : $x_c = x_c.pay\_request$. Après analyse de la politique de confiance, les réactions possibles correspondent aux événements : pay_TTP, pay_ignore.
– La stratégie du client sélectionne l'événement pay_TTP et le prend en compte dans la session. Le contenu de la session courante de l'historique du client est : $x_c = \{object\_request, high\_risk, pay\_request, pay\_TTP\}$. Un message est envoyé au tiers de confiance pour confirmer le paiement au marchand.

Étape 4, côté marchand

– Le marchand reçoit un message du tiers de confiance qui l'informe que le client lui a bien confié la somme correspondant au montant de l'achat. Le marchand ajoute à son historique l'événement $pay\_TTP\_notified$, $x_m = x_m.pay\_TTP\_notified$. La politique du marchand permet maintenant l'envoi de la marchandise, l'événement $object\_send$ est ajouté à l'historique. La session courante de l'historique est :

$$x_m = \{object\_order, average\_risk, ask\_payment, pay\_TTP\_notified, object\_send\}$$

Étape 5, côté client

– Le client reçoit la marchandise et ajoute à son historique l'évenement $object\_received$, $x_c = x_c.object\_received$. Après la vérification de la politique, les candidats possibles sont $pay\_confirm$ et pay_cancel.

– La stratégie de négociation sélectionne le pay_cancel. La session courante est :

$$H_c = \{object\_request, high\_risk, pay\_request, pay\_TTP, object\_received, pay\_cancel\}.$$

Il est possible à cette étape de mettre fin à la négociation.

Étape 6, côté marchand

– Le marchand observe l'événement TTP_time_out, c'est une possibilité de fin de négociation. Celui-ci est pris en compte dans l'historique, $x_m = x_m.TTP\_time\_out$. Pourtant, il n'a pas encore atteint son objectif ($TTP\_confirm$). Il sollicite une re-négociation.

– Le marchand relance le message de l'étape précédente pour re-négocier et obtenir une autre réaction de la part du client, le message correspondant est construit à partir de l'événement $object\_send$. Ce second envoi de $object\_send$ doit être interprété en message du type de *négociation* ($c.f.$6.3.4)

Étape 7, côté client

– Le client reçoit une nouvelle fois l'événement $object\_send$. Cela lui indique que le marchand veut re-négocier l'étape 5. Il dispose d'une autre possibilité de réaction : $pay\_confirm$.

– Si il est honnête, il réagit avec l'événement $pay\_confirm$. Le message de confirmation de paiement est envoyé au tiers de confiance. La session courante x_c de son historique est :

$$x_c = \{object\_request, high\_risk, pay\_request, pay\_TTP, object\_received, pay\_confirm\}.$$

Le client, ayant accepté de négocier, supprime le pay_cancel de son historique.

Étape 8, côté marchand

– Le marchand observe l'événement $pay\_TTP\_received$ qui indique qu'il peut récupérer l'argent auprès du tiers de confiance. Son objectif est atteint.

– La session courante de son historique est :

$$x_m = \{object\_order, average\_risk, ask\_payment, pay\_TTP\_notified, object\_send, pay\_TTP\_received\}$$

Il se trouve dans une situation ou il peut mettre fin à la négociation

Fin de négociation

– Le marchand observe le paiement de l'achat et ajoute l'événement *pay_TTP_received* son historique. Il ne lui reste plus qu'à évaluer la transaction pour clore la session. L'évaluation est saisie interactivement par l'acteur humain du système. En observant tout ce qui s'est passé au long de cette transaction, le vendeur peut mettre une évaluation neutre. L'événement *neutral* sera ajouté à son historique.

$$x_m = \{object\_order, average\_ris, ask\_payment, pay\_TTP\_notified, object\_send, pay\_TTP\_received, neutral\}.$$

– Le client évalue lui aussi la transaction pour terminer la transaction :

$$x_c = \{object\_request, high\_risk, pay\_request, object\_received, pay\_confirm, positive\}$$

6.5.6 Conclusion sur le scénario

Avec la structure d'événements, la politique et la stratégie de négociation de chaque partie impliquée dans la négociation (le client et le vendeur), nous avons montré qu'il était possible d'atteindre les objectifs que nous nous étions fixés :

– la négociation entre les deux parties doit être de *confiance* (respecter les politiques des parties en présence) pour pouvoir mener la transaction à son terme ;
– chaque partie impliquée dans la transaction a son propre objectif dans la négociation : le client paie pour avoir le produit qu'il convoite et le vendeur reçoit le paiement du produit qu'il a livré.
– Le tiers de confiance utilisé pour le paiement dans le scénario est seulement un moyen de traitement. Le processus de négociation et la décision à chaque étape ne concerne que les deux parties engagées dans la transaction. Pour l'instant, nous ne considérons pas les aspects de négociation avec ce tiers de confiance.

6.6 Synthèse

Dans ce chapitre, nous avons présenté notre système de négociation de confiance. Ce système de négociation peut être vu comme un mécanisme engendrant dynamiquement des protocoles respectant les politiques de confiance des participants à une transaction. En appliquant le modèle de gestion de la confiance présenté dans le chapitre 5, nous avons montré qu'il est possible de synthétiser dynamiquement un protocole de communication qui respecte les politiques exprimées par les paires engagées dans la transaction. Nous avons étudié également le scénario d'une application pratique de la négociation pour présenter les aspects concrets de notre système.

Chapitre 7

Gestion du risque

Dans ce chapitre, nous présentons la gestion des risques telle que nous l'avons intégrée dans notre système de gestion de la confiance. Ce modèle de risque est fait pour être appliqué à des transactions du domaine du commerce électronique. Nous avons précédement vu que la prise en compte des risques est un élément très important de ce type d'application, il s'agit d'un des principaux facteurs qui permettent aux différentes entités impliquées de prendre une décision sur la poursuite de la transaction.

Le modèle de risque que nous présentons dans ce chapitre est proche de ceux proposés par Gransidon [36] et le projet SECURE [20]. L'idée principale est de proposer une méthode quantitative pour disposer d'une valeur mesurant le risque et d'utiliser cette valeur pour identifier les différents niveaux de risque utilisables dans nos politques.

Nous présentons d'abord le développement d'un système intégré de la gestion du risque. Ensuite, nous présentons le modèle du risque appliqué dans le contexte de notre système de gestion de la confiance. Enfin, nous donnons un scénario d'application appliquant notre modèle du risque.

7.1 Processus de gestion du risque

Dans cette section, nous présentons brièvement l'état de l'art des outils de gestion du risque. Nous nous intéressons au processus de la gestion du risque pour un système d'information. Cet aspect est abordé dans le guide de management du risque [46]. Nous résumons quelques points importants en précisant l'application des processus de la gestion du risque dans notre système de gestion de la confiance.

7.1.1 Perception du risque

En général, la sensation de risque est un phénomère très subjectif, lié à la perception d'une certaine situation par un invividu donné.

Dans le contexte de notre application (une transaction de commerce électronique), une entité participant à la transaction peut analyser différents facteurs afin de prévenir la situation de risque où son partenaire pourrait se livrer une tricherie en contrefaisant les preuves de paiement ou d'envoi de produit, le mode de contact ou le type de paiement, etc.

7.1.2 Identification du risque

Quand il y a eu perception ou sensation de risque, il faut identifier celui-ci. Il faut reconnaître les signaux relatifs aux risques potentiels importants, aux dangers ou aux conséquences dommageables (par exemple, pertes potentielles). Dans cette étape, on porte l'attention sur les causes (facteur du risque), sur les objets de risque et sur les ressources potentiellement impactées par ces facteurs de risque en regardant les aspects critiques associés à ces éléments.

Dans la cas de notre scénario de transaction du commerce électronique, l'entité peut identifier le risque lié aux pertes dans la transaction : le client a payé mais ne reçoit pas le produit, le vendeur n'a pas reçu un paiement valide après avoir envoyé le produit, etc.

7.1.3 Évaluation du risque

Le but de cette phase est de déterminer le niveau de tous les risques identifiés lors de la phase précédente. Les niveaux dépendent des critères appliqués. En général, nous pouvons identifier les niveaux : *risque élevé, risque moyen ou risque faible.*

Pour l'évaluation, il nous faut prendre en compte l'ensemble des paramères de vulnérabilité : *la cause* (facteur du risque), *l'objet de risque* (l'organisation ou la ressource) et *la conséquence* (l'impact), avec leurs gravités potentielles.

Il y a différentes méthodes pour évaluer le risque, qui dépendent du modèle du risque que nous appliquons au système. Il y a différentes approches pour le modèle du risque dans la littérature comme nous l'avons indiqué dans l'état de l'art (chapitre 3) : *l'approche qualitative, l'approche quantitive ou l'approche hybride.*

En appliquant le modèle du risque, nous obtenons le niveau de risque estimé pour chaque risque identifié. Notre modèle de risque se base sur une approche quantitive, ce qui sera présenté dans la section suivante.

7.1.4 Gestion du risque

Dès que l'on a évalué les plus fortes vulnérabilités, on connaît mieux les causes, les objets de risque, et les conséquences de ces vulnérabilités. Quand on a identifié les risques et leurs niveaux, il convient d'adopter une stratégie pour traiter ces risques afin d'en réduire les différents aspects. Il existe diverses stratégies pour traiter les risques, telles que la prévention, les actions correctives et les palliatifs.

- **Prévention** : l'action consiste à diminuer la probabilité d'occurrence du risque en diminuant ou supprimant certains des facteurs de risque.
- **Préparation de correction** : l'action consiste à diminuer l'effet du risque lorsque celui-ci intervient.
- **Palliatifs - Adaptation au risque** : Le risque peut être considéré comme un événement aléatoire d'une transaction, et la théorie des jeux permet d'associer un coût à ce risque. Il est dès lors possible d'appliquer des stratégies palliatives, qui peuvent avoir comme buts de :
  - *Minimiser les pertes* : il faut rendre l'espérance mathématique des pertes la plus faible possible. Les produits financiers sont des exemples typiques de cette approche.
  - *Maximiser les gains* : il faut cette fois rendre l'espérance mathématique des gains la plus forte possible. Dans cette optique on peut décider d'ignorer les risques.

Dans notre système de gestion de la confiance, nous voulons considérer le risque en tant que facteur de la décision sur la confiance en une action d'une entité. L'idée principale est que nous intégrons le risque à la politique de confiance de l'entité, et cela sera effectué à la prise de décision quand la politique est vérifiée.

Notre modèle de gestion de confiance utilise la structure d'événements en modélisant les actions de l'entité, et en considérant les données échangées avec les autres entités comme des événements. Le niveau de risque sera également considéré comme un événement. Il sera pris en compte dans l'historique des transactions et aussi dans la politique, comme nous l'expliquons dans le chapitre 5. Nous citons ici l'exemple de la politique d'une entité, extrait du scénario du chapitre 5.

$$(\psi_2) : high\_risk \rightarrow (valid\_payment \vee G^{-1}(positive))$$

Cela explicite que si le marchand trouve un risque important pour la transaction, il n'accepte que les commandes dont le paiement est valide, ou si toutes les évaluations des transactions antérieures de ce client sont positives.

7.2 Modèle du risque

Nous appliquons les principes du risque tel que défini en économie, *le risque* lié à une transaction est la perte potentielle qu'elle peut provoquer. Lorsqu'une entité prend une décision concernant une action, il en découle un ensemble de conséquences potentielles et à chacune de ces conséquences peut être associée une probabilité d'occurence.

Notre modèle du risque est une interprétation du modèle théorique *de calcul quantitatif* du risque basé sur la notion de *coût/bénéfice*, que appliquons dans le domaine des applications de e-commerce. Dans notre cas, le bénéfice (ou le coût) d'une transaction est la valeur de gain (ou de perte) de la transaction évaluée par chacune des parties engagée dans la transaction. Le bénéfice indique une transaction réussie, par contre le coût indique une transaction échouée. La transaction peut être réussie pour l'une des parties, mais échouée pour l'autre.

Nous nous intéressons plutôt à l'aspect d'analyse du *coût* des conséquences. Dans la pratique, nous trouvons que réaliser une *action* ou ne pas la réaliser implique toujours un *coût*, et cela peut expliquer le coût lié à des choix différents. Nous donnons un exemple pour concrétiser :

Nous nous intéressons à une action d'adjudication sur un appel d'offre de projet. Si nous ne participons pas, nous perdons le bénéfice que nous pouvons faire dans cette opération. Si nous participons, nous avons des coûts (coût financier, temps de préparation d'une ébauche de solution) et nous perdons cette somme, que nous obtenions ou pas le projet.

Pour les transactions e-commerce, le coût de chaque occurence sera estimé sur la base de la somme et des frais de la transaction. La probabilité d'occurence est déterminée en utilisant l'historique des transactions passées.

Notre modèle de risque utilise la structure d'événements que nous décrivons dans la modélisation du système, et il sera appliqué à notre infrastructure de confiance.

7.2.1 Modèle de calcul *coût/bénéfice*

Le modèle considère le risque associé à l'action d'une entité dans le système. Supposons qu'une entité veuille effectuer une action ; cette action peut avoir un ensemble de

conséquences différentes $o_1, o_2, \ldots o_n$. Chaque conséquence o_i peut se réaliser avec une probabilité p_i. Le *coût* ou le *bénéfice* de chaque *conséquence* sera estimé par l'entité en se basant sur la nature de la conséquence. La probabilité de chaque conséquence p_i sera estimée en utilisant les informations de l'historique des actions passées et du contexte dans lequel se déroule l'action actuelle.

La fonction qui estime le coût ou le bénéfice de l'action sera appliquée à l'ensemble des conséquences et à leurs probabilités. Elle est calculée de la manière suivante :

$$CB = \sum p(o_i) * cb(o_i)$$

Dans cette formule, $p(o_i)$ est la probabilité d'occurence de la conséquence o_i lorsque l'on entreprend l'action et $cb(o_i)$ est le coût ou le bénéfice de cette conséquence.

7.2.2 Modèle de risque appliqué à notre infrastructure de gestion de la confiance

Nous avons présenté notre système de gestion de la confiance aux chapitres 5 et 6. Comme nous l'avons alors indiqué, ce système utilise toutes les sources d'informations disponibles pour prendre ses décisions sur la confiance, et le risque est une des informations que notre système peut utiliser. Nous présentons dans cette section une approche qui permet de déterminer le niveau de risque d'une transaction et de l'intégrer dans ce système de gestion de la confiance.

Chaque l'entité du système gère un historique qui se compose de plusieurs sessions $h = x_1.x_2 \ldots x_n$, où chaque x_i est l'ensemble des événements observés pour cette session. Lorsque l'entité doit prendre une décision concernant l'avancement de la transaction avec son partenaire, le choix qui est fait parmi ceux qui sont possibles s'exprime par un *événement* ou par un ensemble d'événements particuliers. Le coût ou le bénéfice sera déterminé en se basant sur la nature des conséquences liées au choix qui vient d'être fait. Par contre, la probabilité d'occurence de chaque conséquence est estimée grâce aux actions passées que contient l'historique.

7.2.2.1 Coût de la conséquence

La détermination du coût (ou du bénéfice) de chaque conséquence de l'action dépend de la nature de l'action et du contexte spécifique de la transaction.

Dans le contexte d'une transaction du commerce électronique pour un paiement d'un client à un marchand, le coût peut être déterminé avec des paramètres tels que le montant de la transaction, les frais de la transaction et le choix de la méthode de paiement. Nous précisons ces détails dans le scénario d'application de la section 2.

7.2.2.2 Probabilité de la conséquence

Déterminer la probabilité de chaque conséquence peut se faire de manière simple en utilisant l'historique des événements. Chaque action a est représentée par un événement e, qui peut donner un ensemble de conséquences $o_1, \ldots, o_k$. Chaque conséquence directe o_i de l'action a représente également un événement e_i. Pour calculer la probabilité de la conséquence o_i, nous distinguons deux cas :

- *L'historique des transactions est vide*
  Dans ce cas, l'acteur peut proposer un arbre des probabilités représentant sa perception du risque. Pour chaque branche de l'arbre, il peut mettre une valeur appropriée. Nous proposons parfois une distribution d'égalité pour toutes

les conséquences pour simplifier les calculs des premières transactions. Pour la distribution équiprobable, supposons qu'il y a k conséquences directes possibles à partir d'une action initiale, la probabilité de chacune sera $1/k$. Pour les conséquences indirectes, il faut appliquer la loi de dépendance des probabilités expliquée ci-dessous.

– *L'historique de transactions contient des événements*
Pour ce cas, nous pouvons calculer la probabilité basée sur cet historique. Supposons que la conséquence o_i soit représentée par l'événement e_i. Nous déterminons le nombre de sessions de l'historique où nous trouvons cet événement. La probabilité d'occurence de cette conséquence peut être calculée comme le nombre de sessions où nous rencontrons cet événement (nb_{occur}) sur le nombre total de sessions ($N_{session}$) :

$$p_o = nb_{occur}/N_{session}$$

7.2.2.3 Conséquences indirectes et dépendance des probabilités

Nous avons considéré le cas où l'action a, représentée par l'événement e, donne des conséquences directes $o_1, \ldots, o_k$, représentées par les événements $e_1, \ldots, e_k$. Il nous faut aussi considérer les conséquences indirectes d'une telle action. Supposons que l'action a entraîne une conséquence o_i, puis que la conséquence o_i (représentée par l'événement e_i), provoque la conséquence o_{ij} ; nous considérons que o_{ij} est la conséquence indirecte de l'action a. La probabilité de la conséquence o_{ij} depuis l'action a dépend de la probabilité de la conséquence intermédiaire o_i.

Supposons que la probabilité de la conséquence o_i de l'action a est p_i, et que la probabilité de la conséquence o_{ij} en o_i est p_{ij}. Selon la loi de la dépendance des probabilités, la probabilité de la conséquence o_{ij} depuis l'action a sera $p_i * p_{ij}$.

En appliquant cette règle de dépendance des probabilités et la méthode de calcul des probabilités pour les conséquences directes ci-dessus, nous pouvons calculer la probabilité de toutes les conséquences indirectes d'une telle action.

7.2.2.4 Arbre des probabilités

Pour une transaction, il y a plusieurs conséquences. Nous faisons appel à une structure spécifique pour modéliser ces conséquences et leurs probabilités, un arbre, qui utilise des événements comme sommets. Les arcs spécifient les conséquences directes entre les événements et ils sont pondérés par la probabilité de ces conséquences. La racine de l'arbre est l'événement correspondant à l'action intiale de l'interaction. Les feuilles de l'arbre sont les événénements décrivant les conséquences finales de l'action initiale. La figure 7.1 illustre un tel exemple d'arbre.

7.2.3 Complexité du calcul de risque

Nous utilisons l'arbre de probabilité pour modéliser les conséquences et leurs probabilités. Le calcul de probabilité des conséquences nous demande un parcours de tout l'arbre. La complexité de ce problème est très élevée, c'est au moins en $O(2^n)$ en terme de taille de mémoire pour stocker l'arbre et de temps pour le parcourir. Le paramètre n est le nombre d'événements spécifiés dans le système.

Si le système a un grand nombre d'événements, la complexité de calcul du risque avec cet algorithme devient très important.

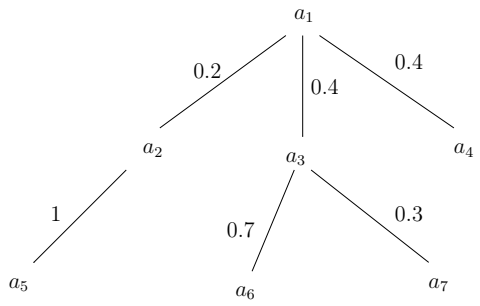


FIG. 7.1 – Un arbre des probabilités

7.2.4 Calcul de la valeur du risque

Pour un risque identifié, nous pouvons calculer la valeur de ce risque grâce au modèle *coût/bénéfice* que nous venons de présenter. La valeur ainsi obtenue sera utilisée pour échantillonner les différents niveaux de risque.

7.2.5 Échantillonnage du risque

Notre historique ne manipulant que des informations logiques, nous devons répartir les valeurs de risque calculées sur une échelle qui dépendra de l'application et de la nature des risques. Les risques peuvent être par exemple répartis sur une échelle avec trois niveaux différents : $high\_risk$, $medium\_risk$ et low_risk, qui indiquent que le risque est respectivement *élevé, moyen ou faible*.

7.2.6 Discussion

La détermination du coût (ou du bénéfice) des conséquences d'une action est importante, car elle donnnera une évaluation approximative du risque. Pour chaque application spécifique, il nous faut proposer une estimation efficace.

Ce modèle de risque est construit en utilisant les approches quantitatives présentées dans la littérature. D'autres approches, qualitatives ou mixtes, pourraient être utilisées pour des applications spécifiques.

7.3 Scénario d'application

Dans cette section, nous présentons le calcul du risque utilisant le modèle présenté pour une transaction de commerce électronique. Le scénario de l'application est celui que nous avons utilisé au chapitre 6. Il s'agit d'une transaction de commerce électronique ; la transaction est paramétrée par le coût de la transaction (la somme totale engagée) et l'historique des entités participant à la transaction.

Dans cet exemple de transaction, la négociation est réalisée entre le client A et le vendeur B. Chaque partie utilise le risque comme paramètre de la transaction. Pour chaque partie, nous identifions d'abord le risque potentiel, puis nous en calculons la valeur en utilisant le modèle que nous venons de présenter. Notons que nous réutilisons la structure d'événements définie dans l'exemple d'application du chapitre 6.

7.3.1 Politique du tiers de paiement

Le client et le vendeur impliqués dans une transaction peut utiliser le service d'un tiers de paiement pour gérer le paiement. Nous supposons que le tiers de paiement TTP utilise pour sa politique par les contraintes suivantes :

- Le client peut déposer un montant pour un paiement d'une transaction, la commission dans ce cas est de 5% du montant de la transaction. Par contre, quand il a déposé le montant pour la transaction, et qu'il veut le récupérer, il doit payer une commission de 20% du montant.
- Le vendeur, pour récupérer le paiement d'une transaction, doit également payer une commission de 5% du montant de la transaction.

7.3.2 Client A

Lors de cette transaction, le client A peut avoir à prendre le risque de *payer le vendeur et de ne pas recevoir l'objet commandé* (perdre la somme d'argent et ne pas avoir le bien convoité). La perte de la somme d'argent, si elle survient, dépend du mode de paiement qui a été utilisé. Le risque que nous déterminons est le risque global pour tout le processus de la transaction. Ce risque est estimé en se basant sur le *coût/bénéfice* des conséquences possibles.

Le client considère que la transaction est réussie s'il reçoit le produit après l'avoir commandé. La transaction réussie est considérée un bénéfice. Nous pouvons quantifier ce bénéfice par un pourcentage du montant du produit. Nous fixons arbitrairement un pourcentage de bénéfice à 10% du montant en cas de transaction réussie.

On choisit de ne pas compter de *coût/bénéfice* au cas où le produit n'est pas obtenu à causedu fait du vendeur, et l'on ne quantifie pas la variation de réputation du client (bon payeur, mauvais payeur, etc - on pourrait considérer qu'une bonne réputation est un bénéfice chiffrable). Pourtant, si le client ne reçoit pas le produit à cause de son comportement (il chosit de ne pas envoyer le paiement), l'on considère ce fait comme perte du client parce que le client a besoin du produit. Cette perte est fixée par une valeur à 10% du montant.

7.3.2.1 Arbre des probabilités

Pour le client, nous obtenons les conséquences possibles pour une transaction, spécifiées par l'arbre des probabilités illustré dans la figure 7.2

7.3.2.2 Spécification des conséquences

Grâce à l'arbre des probabilités, nous pouvons obtenir les conséquences suivantes qui peuvent donner un bénéfice ou causer une perte, concernant l'évaluation du risque. Les conséquences sont données en parcourant l'arbre des probabilités. Les gains et les pertes sont évaluées plus précisémennt à la section 7.3.2.4

- C_1 : A reçoit une demande de paiement après avoir commandé le produit. Il paie directement le vendeur (pay_direct) et reçoit le produit. La perte financière est 0. La transaction est réussie, et le bénéfice est de 10% du montant de la transaction.
- C_2 : A reçoit une demande de paiement après avoir commandé un produit. Il paie directement au vendeur et ne reçoit pas le produit. La perte financière est égale au montant engagé pour la transaction et le client ne dispose pas du produit.

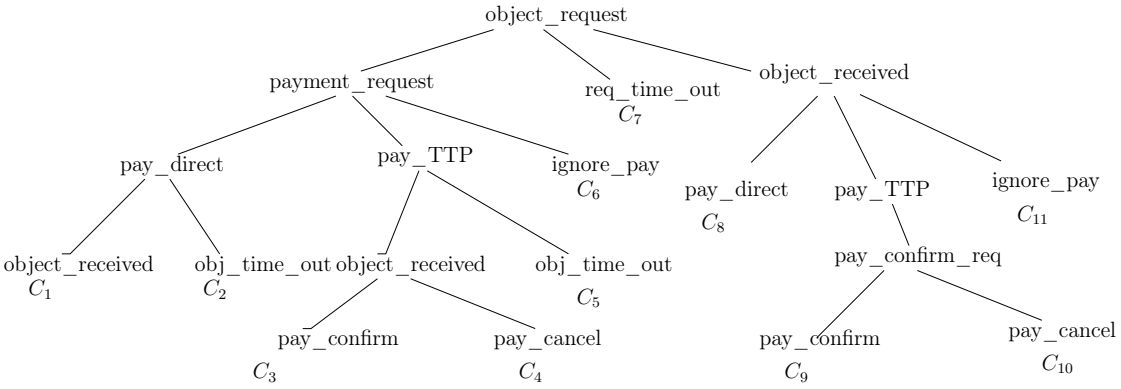


FIG. 7.2 – Arbre des probabilités du client

- C_3 : A reçoit une demande de paiement après avoir commandé le produit. Il paie en utilisant les services d'un tiers de confiance (pay_TTP) et reçoit le produit. Ensuite, il confirme le paiement ($pay\_confirm$). La perte financière est la commission de 5% payée au TTP, selon la politique du TTP, le client dispose du produit et le bénéfice est de 10% du montant.
- C_4 : se déroule comme C_3, sauf à la fin où le client annule le paiement (pay_cancel). Dans ce cas, selon la politique de TTP, il devra payer une commission de 20% du montant de la transaction pour une action de ce type. Il reçoit le produit et le bénéfice est donc de 10% du montant.
- C_5 : A reçoit une demande de paiement après avoir commandé le produit. Il paie en utilisant les services d'un tiers de confiance et ne reçoit pas le produit. La perte financière se monte au montant de la commission (20%) que A verse au tiers de confiance pour obtenir le remboursement du montant de la transaction ; A ne dispose pas du produit.
- C_6 : le client reçoit une demande de paiement après avoir commandé un produit. Il ne paie pas et le client ne dispose pas du produit. La perte dans ce cas est de 10% du montant.
- C_7 : le client commande le produit, mais le vendeur ne valide pas sa commande. La perte est 0.
- C_8 : le client reçoit le produit après l'avoir commandé. Il reçoit ensuite la demande de paiement du vendeur et il le paie par une méthode *direct*. La perte financière dans ce cas est 0. Il dispose du produit et le bénéfice est donc 10% du montant.
- C_9 : le client reçoit le produit après l'avoir commandé. Il reçoit ensuite la demande de paiement du vendeur et il le paie par une méthode utilisant un tiers sûr. Ensuite, il envoie la confirmation de paiement ($pay\_confirm$). Il doit payer une commission de 5% qui est considérée comme une perte. Il dispose du produit et un bénéfice de 10% du montant.
- C_{10} : se déroule comme C_8, sauf à la fin où le client annule le paiement (pay_cancel). Il doit payer une commission de 20% pour récupérer le dépôt selon la politique du TTP. La perte financière dans ce cas est 20% du montant de transaction. Il dispose du produit.
- C_{11} : le client reçoit une demande de paiement après avoir commandé et reçu le produit. Il reçoit ensuite la demande de paiement de vendeur et il ne le paie pas.

Le client, comme pour C_{10} a fait un bénéfice illégal correspondant à la valeur du produit.

En calculant le coût/bénéfice en pratique, si c'est une perte, nous mettons une valeur négative pour la conséquence ; si c'est un bénéfice, nous mettons une valeur positive. Soit P est le coût du produit, nous avons le tableau 7.1 indiquant le coût/bénéfice de l'ensemble de conséquences.

Conséquence	Avant transaction	Après la transaction	Coût/Bénéfice
C_1	P	P*(1+10%)	10%
C_2	P	0	-100%
C_3	P	P*(1-5%+10%)	5%
C_4	P	P*(1+1-20%+10%)	90%
C_5	P	P*(1-20%)	-20%
C_6	P	P*(1-10%)	-10%
C_7	P	P	0%
C_8	P	P*(1+10%)	10%
C_9	P	P*(1-5%+10%)	5%
C_{10}	P	P*(1+1-20%+10%)	90%
C_{11}	P	P*(1+1+10%)	110%

TAB. 7.1 – Conséquences de transactions du client

7.3.2.3 Probabilité des conséquences

Nous devons maintenant déterminer la probabilité, le coût ou le bénéfice de chacune des conséquences potentielles pour évaluer le risque lié à la transaction. Pour la probabilité des conséquences, nous devons considérer deux situations :

1. le client A mène sa première transaction avec le vendeur B. L'historique H est donc limité à la session en cours. Dans ce cas, le client peut proposer un arbre des probabilités en accord avec sa perception. Supposons qu'il propose un arbre où soit associé aux branches qui mènent aux nœuds *payement_ request, req_ time_ out* et *object_ received* les probabilités 0.7, 0.1 et 0.2. Les autres branches de l'arbre sont équiprobables.

   En appliquant l'algorithme de calcul des probabilités de dépendance ci-dessus, nous avons des probabilités de conséquences $C_i(\ i = 1 \ldots 11)$ qui sont respectivement : 0.7/6, 0.7/6, 0.7/12, 0.7/12, 0.7/6, 0.7/3, 0.1, 0.2/3, 0.2/6, 0.2/6, 0.2/3.

2. Le client A a déjà mené des transactions avec le vendeur B. Nous déterminons la probabilité d'occurence de chacune des conséquences en utilisant les informations disponibles dans l'historique. Il faut aussi souligner que le nombre de sessions de l'historique doit être suffisant pour un calcul efficace.

   Nous appliquons l'algorithme de calcul des probabilités pour les conséquences indirectes et nous obtenons la probabilité de chaque conséquence finale abordée ci-dessous. Par exemple, pour calculer la probabilité de la conséquence C_1, nous pouvons réaliser les étapes suivantes :

– Compter le nombre de sessions de l'historique qui contiennent la chaîne des événements suivants dans cet ordre :
{object_ request, payment_ request, pay_ direct, object_ received}, noté par nb_{C_1}.
– La probabilité de cette conséquence sera nb_{C_1}/nb_S. Nous pouvons voir facilement que ce calcul correspond à l'algorithme de calcul des probabilités dépendantes donné ci-dessus.

Le calcul de la probabilité des autres conséquence est réalisé de la même manière. Supposons qu'au moment de la transaction A ait l'historique suivant :

$$
\begin{aligned}
H = &\{object\_request, payment\_request, pay\_direct, low\_risk, object\_received\} \\
&\{object\_request, payment\_request, pay\_direct, low\_risk, time\_out\} \\
&\{object\_request, payment\_request, pay\_direct, low\_risk, time\_out\} \\
&\{object\_request, payment\_request, pay\_TTP, high\_risk, object\_received, pay\_confirm\} \\
&\{object\_request, payment\_request, pay\_TTP, low\_risk, object\_received, pay\_cancel\} \\
&\{object\_request, payment\_request, pay\_TTP, high\_risk, time\_out\} \\
&\{object\_request, object\_received, pay\_direct, high\_risk\}
\end{aligned}
$$

En appliquant l'algorithme de calcul des probabilités, nous obtenons les probabilités p_i de chaque conséquence C_i : $p_1 = 1/7$, $p_2 = 2/7$, p_3=$1/7$, p_4=$1/7$, p_5=$1/7$, $p_6 = p_7 = 0$, $p_8 = 1/7$, $p_9 = p_{10} = p_{11} = 0$.

7.3.2.4 Coût/bénéfice des conséquences

Maintenant, il nous faut déterminer le *coût/bénéfice* de chaque conséquence. Supposons que le montant de la transaction considérée soit 100 € et que la commission retenue en cas de remboursement par le tiers de confiance soit de 10%.

En appliquant les calculs du *coût/bénéfice* donnés dans le tableau 7.1, nous obtenons le coût/bénéfice des conséquences C_i $(i = 1 \ldots 11)$ qui sont respectivement : 10, -100, 5, 90, -20, -10, 0, 10, 5, 90, 110

7.3.2.5 Valeur du risque

En appliquant la formule du calcul de risque à cette transaction, nous obtenons la valeur du risque. Considérons aussi deux cas :

1. L'historique au moment de transaction est vide. Le résultat trouvé dans ce cas sera :
$\sum p(o_i) * cb(o_i) = 10 * 0.7/6 - 100 * 0.7/6 + 5 * 0.7/12 + 90 * 0.7/12 - 20 * 0.7/6 - 10 * 0.7/3 + 0 * 0.1 + 10 * 0.2/3 + 5 * 0.2/6 + 90 * 0.2/6 + 110 * 0.2/3 = 1.5$
2. L'historique au moment de transaction est celui donné ci-dessus, dans le risque est alors :
$\sum p(o_i) * cb(o_i) = 10 * 1/7 - 100 * 2/7 + 5 * 1/7 + 90 * 1/7 - 20 * 1/7 - 10 * 0 + 0 * 0 + 10 * .1/7 + 5 * 0 + 90 * 0 + 110 * 0 = -15$

7.3.3 Vendeur B

Pour limiter ses risques financiers, le vendeur n'envoie le produit au client que lorsqu'il en a reçu le paiement. Selon notre modélisation, un paiement valide est soit un paiement direct (*payment_ received*), soit une confirmation de paiement de la part

d'un tiers de confiance (*pay_TTP_notified*). Lors d'un paiement direct, le vendeur ne prend aucun risque : il peut vérifier que le paiement a été réalisé (que le montant correspondant à la transaction lui a été crédité) avant d'envoyer le produit. Dans la deuxième situation, il est possible que le vendeur ne reçoive pas le paiement dans les situations suivantes : le client ne confirme pas la livraison de sa commande auprès du tiers de confiance ; le tiers de confiance est défaillant et n'effectue pas le paiement.

Le risque considéré est aussi global, basé sur le *coût/bénéfice* des conséquences possibles, en comptant également la commission payée pour le tiers de paiement et le bénéfice en cas d'une transaction réussie (transaction pour laquelle, le vendeur reçoit un paiement valide). Nous fixons aussi ce bénéfice à 10% du montant de la transaction.

Comme dans le cas du client, on considère qu'une vente qui échoue n'entraîne ni gain ni perte, et l'on ne tient pas compte des implications liées à la réputation du vendeur.

7.3.3.1 Arbre des probabilités

L'arbre des probabilités qui modélise ces conséquences possibles d'une transaction du vendeur est illustré dans la figure 7.3

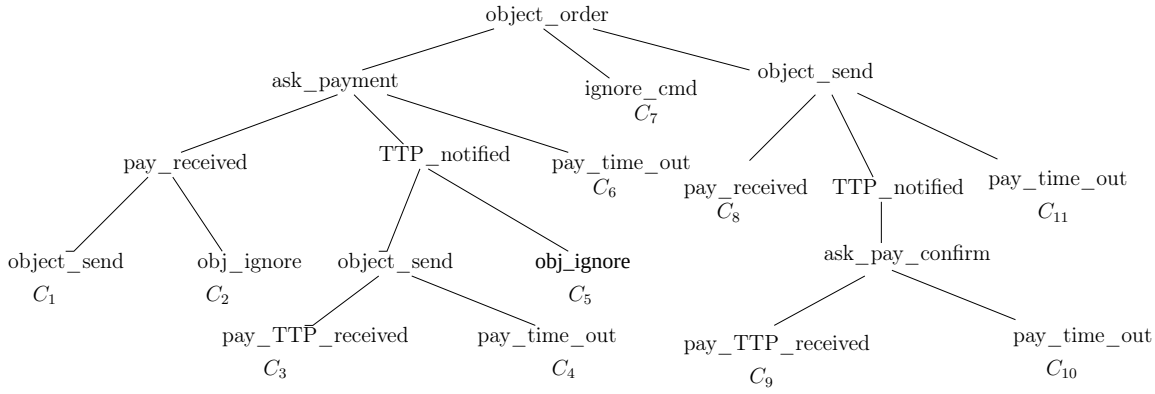


FIG. 7.3 – Arbre des probabilités du vendeur

7.3.3.2 Spécification des conséquences

Nous pouvons trouver les différentes conséquences de la transaction en parcourant l'arbre des probabilités des conséquences.

- C_1 : B reçoit une commande (*object_order*) et il demande un paiement au client. Il envoie le produit au client A après avoir reçu le paiement (*payment_received*) ; la perte financière potentielle est nulle et le produit est envoyé. Le bénéfice est 10% du montant.
- C_2 : B reçoit une commande (*object_order*) et il demande un paiement au client. Il n'envoie pas le produit au client A après avoir reçu le paiement (*payment_received*) ; dans ce cas, il fait un bénéfice illégal qui a pour valeur le montant de la transaction et le produit n'est pas envoyé. Il compte aussi le bénéfice de 10% lié à la réception du paiement.
- C_3 : B reçoit une commande (*object_order*) et il demande un paiement au client. Il envoie le produit au client A après avoir reçu la notification du paiement

(*payment_TTP_notified*). Au final, il reçoit la confirmation du paiement par le tiers (*payment_TTP_received*). La perte financière est de 5% de la valeur du produit pour récupérer le paiement, selon la politique du TTP. La transaction réussit et il a aussi un bénéfice de 10% du montant. Le produit est envoyé.

- C_4 : B reçoit une commande (*object_ order*) et il demande un paiement au client. Il envoie le produit au client A après avoir reçu la notification du paiement (*payment_TTP_notified*) ; Au final, il ne reçoit jamais la confirmation de paiement par le tiers (*payment_tim_out*). La perte dans ce cas est le montant de la transaction.

- C_5 : B reçoit une commande (*object_ order*) et il demande un paiement au client. Il décide ensuite qu'il ne veut plus réaliser la transaction et l'annule donc (*ignore*). Le coût dans ce cas est 0.
- C_6 : B reçoit une commande (*object_ order*) et il demande un paiement au client, le client ignore la transaction. La perte dans ce cas est 0.
- C_7 : B reçoit une commande (*object_ order*) mais il ignore cette commande *ignore_ cmd*. Le coût est égal à 0.
- C_8 : B envoie le produit au client après avoir reçu une commande (*object_ order*). Ensuite il demande un paiement et finalement reçoit un paiement direct valide (*payment_ received*). La perte dans ce cas est 0. La transaction est réussie et il a donc fait un bénéfice de 10% du montant.
- C_9 : B envoie le produit au client après avoir reçu une commande (*object_ order*). Ensuite il demande un paiement et finalement reçoit la notification du paiement par le tiers, *payment_ TTP_ notified*, ensuite une confirmation de paiement (*payment_ received*). La perte financière dans ce cas est 5% du montant de transaction selon la politique du TTP. Le bénéfice est 10% pour ce cas de transaction réussie.
- C_{10} : comme C_8 sauf au final, le vendeur n'a pas la confirmation de paiement (*payment_ TTP_ time-out*). La perte dans ce cas est le montant de la transaction.
- C_{11} : B envoie le produit au client après avoir reçu une commande (*object_ order*). Ensuite il demande un paiement, mais il ne le reçoit jamais (*payment_ time-out*). La perte est égale à la valeur de l'objet.

Soit P le coût du produit, nous avons le tableau 7.2 indiquant le coût/bénéfice de l'ensemble de conséquences.

7.3.3.3 Probabilités des conséquences

Pour déterminer la probabilité de chaque conséquence, nous devons également considérer deux cas : au moment de la transaction, le vendeur fait la première transaction avec le client ou la n^{ieme} transaction.

1. le vendeur B en est à sa première transaction avec A. Dans ce cas, l'historique H de B est vide. Comme dans le cas du client, le vendeur peut proposer un arbre des probabilités selon sa perception pour les premières transactions. Supposons qu'il propose l'arbre, où les probabilités des branches de la racine *ask_ payement, ignore_ cmd* et *object_ send* sont respectivement 0.8, 0.1 et 0.1. Les autres branches de l'arbre sont équiprobables. En appliquant l'algorithme de calcul des probabilités de dépendance ci-dessus, nous obtenons les probabilités des conséquences

Conséquence	Avant transaction	Après la transaction	Coût/Bénéfice
C_1	P	P*(1+10%)	10%
C_2	P	P(1+1+10%)	110%
C_3	P	P*(1-5%+10%)	5%
C_4	P	0	-100%
C_5	P	P	0%
C_6	P	P	0%
C_7	P	P	0%
C_8	P	P*(1+10%)	10%
C_9	P	P(1-5%+10%)	5%
C_{10}	P	0	-100%
C_{11}	P	0	-100%

TAB. 7.2 – Conséquences de transactions du vendeur

$C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}$ *et* C_{11} qui sont respectivement : 0.8/6, 0.8/6, 0.8/12, 0.8/12, 0.8/8, 0.8/3, 0.1, 0.1/3, 0.1/6, 0.1/6 et 0.1/3.

2. le vendeur B a déjà effectué des transactions avec A. Dans ce cas, la probabilité des conséquences C_i sera déterminée en utilisant les informations disponibles de l'historique des transactions.

   De la même façon, nous appliquons l'algorithme de calcul des probabilités pour les conséquences indirectes. Pour calculer la probabilité de la conséquence C_1, nous pouvons réaliser les étapes suivantes :
   - Compter le nombre de sessions de l'historique qui contiennent la chaîne des événements suivante : *{object_ order, ask_ payment, payment_ received, object_ send}, noté par* nb_{C_1}.
   - La probabilité de cette conséquence sera nb_{C_1}/nb_S.

   La probabilité des autres conséquence sera calculée de la même manière. Supposons qu'au moment de la transaction B, ait l'historique suivant :

$$
\begin{aligned}
H_S = &\{object\_order, ask\_payment, pay\_received, object\_send \dots\} \\
&\{object\_order, ask\_payment, pay\_TTP\_notified, ignore \dots\} \\
&\{object\_order, ask\_payment, pay\_TTP\_notified, object\_send, payment\_received \dots\} \\
&\{object\_order, ask\_payment, pay\_TTP\_notified, object\_send, payment\_received \dots\} \\
&\{object\_order, ask\_payment, pay\_TTP\_notified, object\_send, payment\_time\_out \dots \\
&\{object\_order, object\_send, ask\_payment, payment\_received \dots\} \\
&\{object\_order, object\_send, ask\_payment, payment\_time - out \dots\}
\end{aligned}
$$

   En appliquant l'algorithme de calcul de la probabilité des conséquences, nous obtenons la probabilité p_i de chaque conséquence C_i : p_1=1/7, p_2=0, p_3=2/7, p_4=1/7,$p5$=1/7, p_6=0, p_7=0, p_8=1/7, $p_9 = p_{10} = 0$, p_{11}=1/7.

7.3.3.4 Coût/bénéfice

Supposons que le montant de la transaction considérée soit 100 €. Nous pouvons estimer le *coût/bénéfice* de chaque conséquence en utilisant le tableau de

calcul *coût/bénéfice* 7.2. Nous obtenons le coût/bénéfice des conséquences C_i ($i = 1 \ldots 11$) qui sont respectivement : 10, 110, 5, -100, 0, 0, 0, 10, 5, -100, -100.

7.3.3.5 Évaluation

La valeur du risque est évaluée avec la probabilité et le *coût/bénéfice* de chaque conséquence estimée, en utilisant la formule de la fonction *PDF*. Considérons deux cas :

(a) L'historique au moment de la transaction est vide. Le résultat trouvé dans ce cas sera :
$\sum p(o_i) * cb(o_i) = 10 * 0.8/6 + 110 * 0.8/6 + 5 * 0.8/12 - 100 * 0.8/12 + 0 * 0.8/8 + 0 * 0.8/3 + 0 * 0.1 + 10 * 0.1/3 + 5 * 0.1/6 - 100 * 0.1/6 - 100 * 0.1/3 = 5.1$

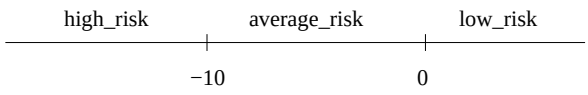
(b) L'historique au moment de transaction est celui donné ci-dessous, dans ce cas le risque est :
$\sum p(o_i) * cb(o_i) = 10 * 1/7 + 110 * 0 + 5 * 2/7 - 100 * 1/7 + 0 * 1/7 + 0 * 0 + 0 * 0 + 10 * 1/7 + 5 * 0 - 100 * 0 - 100 * 1/7 = -24$

7.3.4 Échantillonnage du risque

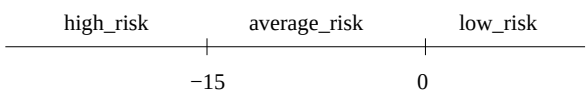
Nous venons de voir de quelle manière peut être calculée la valeur du risque pris par chacune des parties engagées dans une transaction. Il est nécessaire de ré-échantilloner ces niveaux de risque pour pouvoir les traduire en événements : low_risk, $medium\_risk$, $high\_risk$.

Les seuils pour l'échantillonnage peuvent être calculés après le processus d'apprentissage par les essais. Le bénéfice est une valeur positive, et le coût porte une valeur négative dans ce cas. Dans le cadre du sénario proposé en exemple, après quelques essais d'apprentissage de chaque côté, le client et le vendeur, les échelles suivantes peuvent être appliquées :

- Pour le client, nous distinguons les niveaux :
  - ≥ 0 : *low_ risk*
  - *-10 à 0 : average_ risk*
  - $\leq$ *-10 : high_ risk*



- Pour le vendeur, nous distinguons les niveaux :
  - ≥ 0 : *low_ risk*
  - *-15 à 0 average_ risk*
  - $\leq$ *-15 : high_ risk*



7.3.5 Discussion

Quand nous identifions le risque de la transaction, nous nous intéressons aux risques importants qui concernent la réussite globale de la transaction, ceux que nous avons

abordés ci-dessus. Pourtant, il y a aussi d'autres aspects également liés aux risques que nous ne prenons pas en compte, tels que :

- Pour le client, il y a un risque que la qualité du produit ne soit pas conforme à celle qui était décrite.
- Pour le vendeur, il y a un risque que le paiement du produit livré ait été fait avec un moyen de paiement volé ou falsifié (numéro de carte VISA volé).
- Il y a aussi des risques liés au transport entre les deux parties, tels que par exemple la perte du produit, un retard de livraison...
- Les coûts indirects liés à la baisse de réputation d'un client ou d'un vendeur, etc.

La prise en compte de ces aspects aurait alourdi l'exemple proposé, mais elle peut s'effectuer si l'on veut obtenir des modèles plus réalistes.

Nous pouvons constater facilement que la valeur du risque évolue après chaque transaction. Le fait que l'historique change après chaque transaction implique le changement de la probabilité des conséquences, et le risque évolue donc. C'est aussi un fait indiquant qu'il est nécessaire de réévaluer les probabilités quand on veut lancer une nouvelle transaction.

7.4 Synthèse

Nous avons présenté dans ce chapitre notre réalisation de la gestion du risque. Nous avons défini des phases dans la gestion du risque afin de concevoir ce module. Ensuite, un modèle du risque basé sur l'approche quantitive a été intégré à notre système de gestion de la confiance. Enfin, nous avons détaillé notre modèle en analysant le risque pour le scénario d'application d'une transaction du commerce electronique.

Chapitre 8

Prototype de négociation de la confiance

Dans ce chapitre, nous présentons l'implémentation de notre prototype de bibliothèque de négociation de la confiance. Il s'agit d'un prototype qui implémente le scénario d'application pour les transactions de e-commerce. Le modèle de la négociation de ce prototype a été présenté au chapitre 6 et se compose de plusieurs éléments : l'historique, la politique de confiance et la vérification de la conformité de cette politique. Nous avons choisi de réaliser ce prototype sous la forme d'un module de négociation de manière à ce qu'il puisse être facilement intégré à différentes applications.

Dans un premier temps, nous allons présenter l'architecture générale du prototype, puis nous présenterons les détails de l'implémentation de chaque module.

Le prototype est implémenté en langage Java. Dans le cadre du développement de ce prototype, nous utilisons certains *packages* et certaines classes proposés par K. Krukow [58] avec son application JavaHBAC[1].

8.1 Architecture

La négociation entre deux entités est réalisée sous la forme d'un protocole mis en œuvre par le module intégré à chaque l'entité. Le prototype est construit pour implémenter le scénario de négociation entre les deux entités. Il utilise quelques éléments (classes, une partie du schéma XML) issus de l'application JavaHBAC [58].

Le prototype est implémenté en Java. Chaque partie participant à la négociation, le client ou le vendeur, est représentée par une application. La communication entre deux parties (les deux applications) est assurée par le protocole de négociation. L'architecture commune à chaque application est présentée figure 8.1. Elle est constituée des composants suivants :

- Policy : la spécification de la politique de chaque participant est stockée sous la forme d'un fichier XML. Ce formalisme nous permet de représenter et de manipuler efficacement les politiques de confiance (PP-LTL). Ce fichier de politique est chargé et traduit dans la représentation interne au programme à l'aide d'un analyseur syntaxique XML standard lors de l'initialisation du module.
- Negociation Agent : ce module gère la stratégie de l'acteur associé à l'application. Il s'agit de classes Java qui sont en relation avec les composants suivants : *History,*

[1] l'application JavaHBAC est un démonstrateur de son système de réputation qui permet de contrôler les droits d'accès à des fichiers disponibles sur une machine.

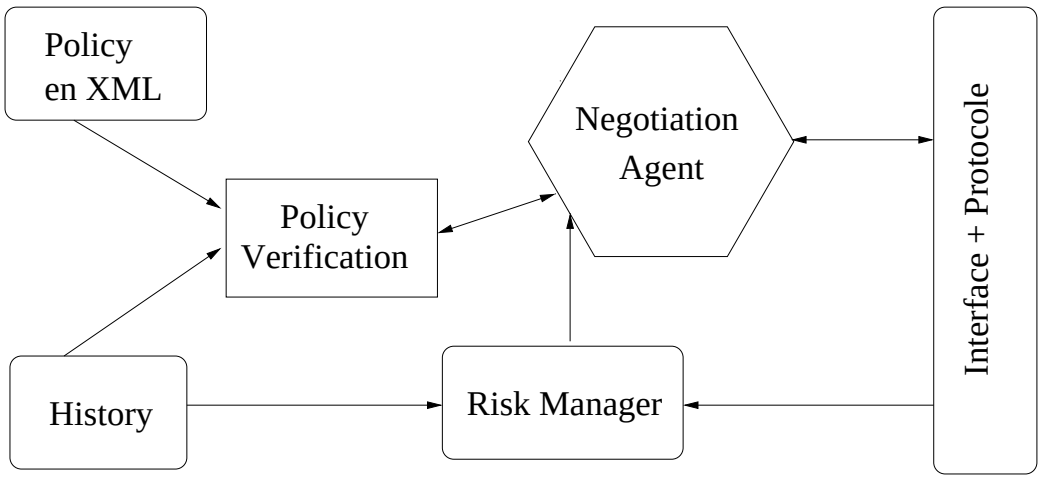


FIG. 8.1 – Architecture de l'application

Risk Manager, Interface & Protocole et Policy Verification.

- Risk Manager : ce module réalise les calculs de risque conncernant la transaction en cours.
- Policy verification : ce module est chargé de vérifier en permanence que la politique de confiance est satisfaite (vérification de la satisfiabilité d'une formule logique PP-LTL sur d'un historique donné). Il doit pouvoir accéder à l'historique et à la politique de l'acteur.
- Interface & Protocole : ce composant assure deux fonctions — l'échange d'informations entre les deux parties et l'interface interactive avec l'application. L'interface permet à l'utilisateur d'interagir avec le module lorsque lui seul est à même de prendre une décision, et elle permet aussi d'afficher des informations ou des explications sur la transaction en cours. Les messages échangés par les deux parties sont pré-définis et constituent le protocole de communication.

La description de l'architecture de l'application nous a donné une vue globale du prototype. Le développement de ces composants et leur fonctionnement est détaillé dans les sections suivantes.

8.2 Spécification de la politique en XML

La politique de confiance de chaque partie est exprimée par une formule logique PP-LTL telle que celle que nous avons présentée dans le scénario d'application du chapitre 6. Pour faciliter la spécification et la manipulation des politiques par le programme, nous utilisons le format XML pour les sauvegarder. De cette façon, la politique de chaque entité est stockée dans un fichier XML qui utilise un schéma pré-défini. Dans JavaHBAC [58], l'auteur a proposé un schéma DSD (Document Structure Description) comportant les éléments de base indispensables à la spécification de la politique d'une application gérant le contrôle d'accès des fichiers. Nous étendons ce schéma pour pouvoir l'utiliser lors de la spécification de la politique de notre scénario.

La politique de confiance de chaque entité est indépendante du module de gestion de la confiance car nous devons pouvoir la modifier (correction, ajout ou suppression de conditions). Les API du langage Java offrent aujourd'hui un accès à des analyseurs

syntaxiques de XML, de nombreux outils standards comme `xmllint` ou `xsltproc` permettent de vérifier la syntaxe ou de transformer des documents XML, et comme de plus ce format de document est facile à partager, la spécification de la politique au format XML nous est apparue être un bon choix.

8.2.1 Schéma DSD

Le schéma de description de document proposé par A.Moller et al [56, 69] dans le cadre du projet DSD2.0 nous fournit les éléments de base pour la représentation de nos politiques en logique PP-LTL. Nous pouvons définir les éléments dont nous avons besoin (*politiques, actions, formules, événements, quantificateurs*) à partir d'un *tag* « element ».

En définissant la politique en DSD, nous pouvons adapter *l'analyseur syntaxique* fourni dans le prototype JavaHBAC [58] pour développer notre application. L'exemple suivant représente une déclaration de la politique en utilisant la syntaxe DSD.

```
<if>
    <element name="p:policy"/>
    <declare>
        <contents>
            <repeat>
                <sequence>
                    <element name="p:actions"/>
                    <element name="p:behaviour"/>
                </sequence>
            </repeat>
            <normalize whitespace="trim"/>
        </contents>
    </declare>
</if>
```

Pour appliquer ce schéma à notre application, il nous faut l'étendre en ajoutant les éléments qui définissent les différents types de permissions pour chaque événement possible dans les formules de la politique.

– les événements concernant le client

```
<contenttype id="p:action">
    <union>
        <string value="object_request"/>
        <string value="object_received"/>
        <string value="obj_time_out"/>

        <string value="payment_request"/>
        <string value="pay_direct"/>
        <string value="ignore_pay"/>
        <string value="pay_confirm"/>
        <string value="pay_cancel"/>
    </union>
</contenttype>
```

– les événements concernant le vendeur

```
<contenttype id="p:action">
    <union>
        <string value="object_order"/>
        <string value="object_send"/>
        <string value="ignore_cmd"/>

        <string value="ask_payment"/>
        <string value="payment_received"/>
        <string value="payment_TTP_received"/>
        <string value="pay_TTP_notified"/>
        <string value="payment_time_out"/>
        <string value="pay_confirm_TTP"/>
        <string value="payment_time_out"/>
    </union>
</contenttype>
```

La définition des éléments de base en DSD pour la politique telles que les actions, les opérations logique, les formules... se trouve dans l'annexe A.

8.2.2 Politiques des participants

Nous avons défini la politique de chacune des parties engagées dans l'application du scénario sous la forme d'une formule logique PP-LTL. Cette section présente comment nous allons représenter ces politiques en XML, en respectant le schéma que nous venons de décrire.

Client

La politique du client est donnée par les formules logiques suivantes :
- $\psi_1 : pay\_direct \Longrightarrow \neg(payment\_request \wedge high\_risk \wedge F^{-1}(pay\_direct \wedge obj\_time\_out))$
- $\psi_2 : pay\_TTP \Longrightarrow payment\_request \wedge [low\_risk \vee (average\_risk \wedge G^{-1}(pay\_direct \wedge obj\_time\_out)]$

La représentation de la politique ψ_1 en format XML, certains éléments XML définissent les événements pris en compte, d'autres représentant les règles, avec une représentation arborescente classique des formules logiques :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<policy wmlns:xi="http://www.w3.org/2001/XInclude"
        xmlns="/dsd/quantifiedjavapolicies">
    <actions>
        <negopermission>payment_request</negopermission>
        <negopermission>object_received</negopermission>
                .............
        <negopermission>payment_direct</negopermission>
        <negopermission>payment_time_out</negopermission>
    </actions>
    <behaviour>
        <implication>
```

```
            <premise>
                <event>pay_direct</event>
            </premise>

            <conclusion>
                <not>
                    <and>
                        <event>payment_request</event>
                        <event>high_risk</event>
                        <sometime>
                            <and>
                                <event>pay_direct</event>
                                <event>obj_time_out</event>
                            </and>
                        </sometime>
                    </and>
                </not>
            </conclusion>

        </implication>
    </behaviour>
</policy>
```

Représentation de la politique ψ_2 du client en format XML est donnée en annexe A.

Vendeur

Nous traitons de manière similaire la politique du vendeur, décrite au chapitre 6. La représentation XML complète de cette politique est donnée en annexe A.

8.2.3 Analyseur syntaxique

Un analyseur de fichiers de politique au format XML est fourni avec l'application JavaHBAC. Nous avons adapté les classes *PolicyParser* et *QPolicyParser* à notre application en y en ajoutant la prise en compte des actions et des évéments propres à notre scénario.

8.3 Réalisation du module de vérification

À la date de rédaction de ce manuscrit, nous réalisons les classes de vérification de politique proposées dans le cadre de l'application JavaHBAC [58] en utilisant les *packages* de vérification de modèles *Automaton* et *JavaBDD*, disponibles en ligne.

Une autre ébauche du module de vérification de la politique de sécurité est réalisée dans le cadre du stage de Master de recherche de Nancy Betancourt [7]. La vérification de la politique dans cette approche se base sur le problème de la satisfaction des formules logiques. L'implémentation utilise le solver MiniSAT développé par Niklas Eén et Niklas Soresson [21]. Cette implémentation doit encore être améliorée pour pouvoir être utilisable.

8.4 Calcul du risque

Ce module est développé sous la forme d'une classe Java (*RiskManager.java*) qui implante l'algorithme de calcul du risque que nous avons présenté au chapitre 7. Ce calcul prend comme paramètres d'entrée l'historique de l'entité et les informations échangées lors de la phase de négociation et fournit comme résultat l'évaluation du risque sous la forme d'un événement. Une partie du code source de ce composant est présentée en annexe.

8.5 Interface et protocole de négociation

Le protocole assure les communications entre deux parties en négociation. Il utilise les *sockets* pour transférer les informations sur le réseau.

Le client et le vendeur, les deux parties en négociation, sont deux programmes : le *client* et le *serveur* respectivement échangent leurs messages sous un *format standardisé*. Quand le programme (*client* ou *serveur*) reçoit un message, il le transmet au module *Event Analyzer* pour que le message soit transformé en événement. Ensuite, cet événement sera transmis comme paramètre à l'*Agent de Négociation* pour réaliser la négociation.

8.5.1 Échanges

Les échanges sont réalisés de façon continue entre les deux programmes.

- Le client envoie un message au serveur, et quand le serveur le reçoit, il appelle la méthode *process()* (côté serveur) de l'agent de négociation, pour savoir comment réagir ;
- le serveur envoie un message au client et quand le client le reçoit, il appelle lui aussi la méthode *process* (côté client) de l'agent de négociation.

Les messages d'échange sont de types différents : *proposition, négociation, rejet* et *conclusion* qui sont décrits dans la section suivante. Ce processus continu jusqu'à ce que l'une des deux parties échange un message de type *conclusion* qui indique la fin de la négociation. Le squelette des programmes *client* et *serveur* est fourni dans l'annexe B.

8.5.2 Event Analyzer

Cette classe traite les transformations des événements en message et réciproquement. Un message d'entrée est transformé en événement pour être utilisé dans l'*agent de négociation*. À l'inverse, la réponse de l'*agent de négociation* est transformée en message pour permettre l'échange avec l'autre partie à travers le protocole.

```
public class EventAnalyzer{
{
    String eventname[]; // tableau des événements
    Message msg_list[]; // liste de messages
    String event ;      // evennt
    Message msg;

    public EventAnayze() {};
```

```
    public Message EventToMessage(String event){return msg;}
    public Event   MessageToEvent(Message msg){return event;}
}
```

8.5.3 Structure des messages

Comme indiqué, nous avons besoin d'un format précis pour les messages échangés par ce protocole. Le message se compose des informations suivantes : *un identifiant du message, le type du message, l'événement équivalent et les données échangées*. Nous définissons une classe qui permet de stocker ces informations et qui offre les traitements (méthodes) nécessaires à leur utilisation.

```
public class Message implements Serializable
{
    int ident; //identifiant du msg
    int type;  //type du message
    String event //événement équivalent
    String data; //données échangées
};
```

La spécification détaillée de cette classe est donnée dans l'Annexe B.

8.6 Agent de négociation

Ce module traite les échanges de négociation en terme d'événements. C'est un processus parallèle au protocole d'échange. Nous abordons ici les *classes* principales intégrées à ce module.

8.6.1 Processus de négociation

Cette classe (AgentNegotiation.java) gère tout au long de son déroulement le processus de négociation. L'Agent réalise les traitements suivants :

- À chaque étape, il prend comme paramêtre l'événement, transformé à partir du message échangé, pour effectuer la *négociation* de l'étape courante, en créant un nouvel objet de la classe *Étape de négociation*
- Le processus continue ainsi jusqu'à ce qu'il ait reçu un événement, ou qu'il propose un événement, qui corresponde à un message de type *conclusion*, ce qui indique la fin de la négociation.

Le détail de la classe se trouve dans l'annexe B.

8.6.2 Étape de négociation

Cette classe traite la *négociation* à chaque étape du processus. Quand l'entité reçoit un événement spécifiant une proposition du partenaire, elle a besoin de l'analyser pour donner une *proposition, une négociation, un rejet* ou *une conclusion* correspondant. Les opérations suivantes sont traitées dans la classe.

- Calculer E_C, l'ensemble des événements candidats à partir de la structure d'événements. Ce traitement appellera la procédure de vérification de la politique, les classes du module de *Vérification*.
- Appliquer la stratégie (*Strategy*) pour déterminer avec quel événement réagir.

– Envoyer l'événement à la classe « *Protocole et Interface* » pour construire les messages à échanger.

Le squelette général de cette classe est donné dans l'annexe B.

8.6.3 Stratégie de négociation

À chaque étape de la négociation, quand l'entité a un ensemble d'événements candidats E_C avec lesquels elle peut réagir, elle applique sa stratégie pour choisir le meilleur événement. Le fonctionnement de cette classe est basé sur l'algorithme *StrategieNegociation* que nous avons présenté dans le chapitre 6. L'algorithme utilise le *poids de la causalité* des événements candidats, l'événement ayant un poids le plus faible est choisi.

8.6.4 Historique

L'historique des transactions est important pour la négociation. C'est un ensemble de *configurations* où chacune d'elle contient l'ensemble des événements correspondant à une session (une transaction). Nous avons besoin de stocker l'historique de chaque session pour que l'entité puisse s'en servir lors des négociations qu'elle effectuera ultérieurement.

En terme de structure et de données manipulées pour représenter l'historique, nous choisissons dans notre implémentation le format XML pour les stocker. Un description de ce format XML est donné dans l'annexe A.

8.7 Synthèse

Dans ce chapitre, nous avons présenté la structure de l'implémentation de notre prototype de négociation, en utilisant le modèle de négociation présenté au chapitre 6. Le prototype actuel est en cours de réécriture pour obtenir une version complète.

Chapitre 9

Conclusion

Le travail que nous avons présenté dans cette thèse aborde un problème significatif de la recherche en sécurité informatique : le problème de gestion de la confiance. Lors de l'utilisation d'applications Internet (commerce électronique, chat, ...), une question se pose toujours : lorsque nous prenons contact avec un acteur (une personne ou un utilisateur virtuel) que nous ne connaissons pas, comment pouvons-nous lui faire confiance pour réaliser une transaction ? Avec le développement très rapide des applications Internet et leur utilisation pour la réalisation de transactions commerciales de plus en plus importantes (en quantités et en montants), disposer de solutions à ce problème s'avère un point crucial.

La solution proposée est un système de gestion de la confiance qui permet à une entité d'évaluer la confiance qu'elle peut accorder à une autre entité du système.

Le système se base sur un modèle de confiance dans lequel toutes les sources d'informations disponibles peuvent être utilisées pour la décision de la confiance. De plus, le système apporte un mécanisme qui permet d'établir une confiance mutuelle entre deux acteurs tout au long de leur transaction. La confiance est acquise par la négociation.

Ce problème a été abordé de plusieurs manières dans la littérature. Dans cette thèse, nous avons procédé dans l'ordre suivant.

- Nous avons tout d'abord présenté l'état de l'art du problème (chapitre 1). Pour cela, nous avons analysé les notions de base concernant le problème de la confiance. Ensuite, nous avons présenté les systèmes de gestion de la confiance existant permettant une résolution du problème. Nous avons également analysé les avantages et les limites de ces systèmes. Cette discussion a fait l'objet des chapitres 2, 3 et 4.
- Dans une seconde étape, nous avons donné une description globale de notre système de gestion de la confiance. Ensuite, nous avons développé les différents composants de notre système tels que *la prise de décision, le modèle de négociation ou le modèle du risque*. Cette description est réalisée dans les chapitres 5, 6, 7 et 8.

Dans les sections suivantes, nous effectuons un résumé de notre contribution à une solution de ce problème de recherche. Ensuite, nous donnons quelques perspectives pour la continuation des développements réalisés dans le cadre de cette thèse.

9.1 Apports

Dans le cadre de ce travail de thèse, nous avons proposé quelques contributions pour un système de gestion de la confiance.

- **Un modèle de la confiance** : notre approche pour un modèle de la confiance est une approche *hybride*. Dans cette approche, nous utilisons toutes les sources d'informations disponibles tels que les qualifications, les recommandations, la réputation ou le risque, pour prendre une décision sur le fait de faire confiance ou non à une entité. Le modèle proposé utilise une formalisation du système avec la structure d'événements. Dans cette formalisation, toutes les actions, les données échangées (qualification, réputation, recommandation, risque) sont considérées comme des événements. La structure d'événements est une base pour construire les politiques de gestion de la confiance de chaque entité, en utilisant une logique spécifique, la logique PP-LTL. La politique de gestion de la confiance est exprimée comme une formule utilisant cette logique, notée ψ. L'historique H garde toutes les données des événements observés lors de transactions antérieures. La prise de décision sur le fait d'accorder sa confiance ou non à une entité consiste en une vérification de la satisfaisabilité de la politique avec l'historique de cette entité : $H \models \psi$.

- **Un modèle de négociation** : ce modèle est également formalisé par la structure d'événements proposée pour le système. Le modèle formalise la négociation entre deux parties en vue d'établir une confiance mutuelle. Dans ce modèle, chaque entité a sa propre politique de confiance, présentée par la logique LTL et une base de l'historique des interactions avec les autres entités. La négociation entre les deux parties a lieu en plusieurs étapes avec l'aide de composants différents : le protocole de négociation, la stratégie de négociation et la vérification de la politique. Dans ce travail de thèse, ce modèle de négociation est concrétisé par un scénario d'application : une transaction en ligne (vente/achat).

- **Un modèle de risque** : le risque est également un facteur important dans notre système de gestion de la confiance, il est utilisé pour la prise de décision de la confiance. Nous avons proposé un modèle de risque en formalisant le système en structure d'événements. Ce modèle de risque est un modèle qualitatif qui évalue le risque basé sur le calcul *coût/bénéfice* de chaque action. Chaque action peut avoir plusieurs conséquences liées à une probabilité. La valeur du risque est évaluée en combinant toutes ces conséquences et leurs probabilités.

- **Une expérimentation du modèle de négociation** : nous donnons dans cette thèse un prototype de négociation de la confiance. C'est la première expérimentation de notre modèle de négociation. Le prototype implémente le scénario d'application d'une transaction en ligne. Il est développé en Java et la politique de gestion de la confiance est décrite dans un document au format XML. Nous utilisons la technique de programmation réseaux avec des *sockets* pour le protocole de négociation.

9.2 Perspectives

Dans cette thèse, nous avons présenté notre système de gestion de la confiance qui permet d'établir la confiance mutuelle entre les entités dans le système en utilisant toutes les sources d'informations disponibles : les qualifications, les recommandations, la réputation, le risque... Le but de la thèse est atteint mais nous donnons quelques perspectives pour améliorer notre système et la recherche dans ce domaine.

- Proposer un algorithme plus efficace pour la stratégie de la négociation.

- Pour la mise en œuvre, nous pouvons suggérer deux nouveaux composants : un module de stratégie de négociation et une interface pratique pour les utilisateurs. Il serait également intéressant d'améliorer le module de vérification de la politique avec un algorithme plus optimisé.

- Le langage permettant d'exprimer la politique dans notre système utilise la logique PP-LTL qui peut être compliquée à appréhender. Nous pouvons imaginer de développer un langage de description de politique basé sur ce principe et qui peut être utilisé plus facilement dans les applications.

Annexe A

Spécification de la politique

Nous proposons dans cette annexe des extraits de la description complète des politiques décrites dans cette thèse. Le formalisme utilisé est DSD 2.0 [69], assez similaire à XSchema, mais moins lourd à mettre en œuvre.

A.1 Schéma DSD

– Déclaration de la politique

```
<if>
    <element name="p:policy"/>
    <declare>
        <contents>
            <repeat>
                <sequence>
                    <element name="p:actions"/>
                    <element name="p:behaviour"/>
                </sequence>
            </repeat>
            <normalize whitespace="trim"/>
        </contents>
    </declare>
</if>
```

– Déclaration des actions

```
<if>
    <element name="p:actions"/>
    <declare>
        <contents>
            <repeat min="1">
                <contenttype ref="p:negopermission"/>
            </repeat>
        </contents>
    </declare>
</if>
```

– Déclaration des événements

```
  <if><element name="p:event"/>
    <declare>
      <contents>
    <contenttype ref="p:negopermission"/>
        <normalize whitespace="trim"/>
      </contents>
    </declare>
  </if>
```

– Déclaration des formules

```
<contenttype id="p:formula">
    <union>
        <string value="True"/>
        <string value="False"/>
        <element name="p:event"/>
        <element name="p:possible"/>
        <element name="p:and"/>
        <element name="p:not"/>
        <element name="p:last"/>
        <element name="p:since"/>
        <element name="p:sometime"/>
        <element name="p:always"/>
        <element name="p:or"/>
        <element name="p:implication"/>
    </union>
</contenttype>
```

– Déclaration le quantificateur ET.

```
<if>
    <element name="p:and"/>
    <declare>
        <contents>
            <contenttype ref="p:formula"/>
            <normalize whitespace="trim"/>
        </contents>
    </declare>
</if>
```

– Déclaration l'opérateur d'implication.

```
<if>
    <element name="p:implication"/>
    <declare>
        <contents>
            <sequence>
                <element name="p:premise"/>
                <element name="p:conclusion"/>
            </sequence>
            <normalize whitespace="trim"/>
        </contents>
    </declare>
```

```
</if>
```

– les événements concernant le client

```
<contenttype id="p:action">
    <union>
        <string value="object_request"/>
        <string value="object_received"/>
        <string value="time_out"/>
        <string value="payement_request"/>
        <string value="pay_direct"/>
        <string value="ignore"/>
        <string value="pay_confirm"/>
        <string value="pay_cancel"/>
    </union>
</contenttype>
```

– les événements concernant le vendeur

```
<contenttype id="p:action">
    <union>
        <string value="object_order"/>
        <string value="object_send"/>
        <string value="ignore"/>
        <string value="ask_payement"/>
        <string value="payement_received"/>
        <string value="pay_TTP_notified"/>
        <string value="payement_time_out"/>
        <string value="payement_received"/>
        <string value="payement_time_out"/>
    </union>
</contenttype>
```

A.2 La politique du client

Représentation de la politique ψ_2 du client en format XML

$\psi_2 : pay\_TTP \Longrightarrow payment\_request \wedge [low\_risk \vee (average\_risk \wedge G^{-1}(pay\_direct \wedge obj\_time\_out)]$

```
<policy >
    <actions>...</actions>
    <behaviour>
        <implication>
            <premise>
                <event>pay_TTP</event>
            </premise>
            <conclusion>
                <and>
                    <and>
```

```
                    <event>payment_request</event>
                    <or>
                        <event>low_risk</event>
                        <and>
                            <event>average_risk</event>
                            <always>
                                <and>
                                    <event>pay_direct</event>
                                    <event>obj_time_out</event>
                                </and>
                            </always>
                        </and>
                    </or>
                </and>
            </and>
        </conclusion>
    </implication>
</behaviour>
</policy>
```

A.3 La politique du vendeur

1. $\psi_1 : G^{-1}(negative) \Longrightarrow obj\_order \wedge ignore\_cmd)$

```
<policy>
    <actions>...</actions>
    <behaviour>
        <implication>
            <premise>
                <always>
                    <event>negative</event>
                </always>
            </premise>
            <conclusion>
                <and>
                        <event>obj_order</event>
                        <event>ignore_cmd</event>
                </and>
            </conclusion>
        </implication>
    </behaviour>
</policy>
```

2. $\psi_2 : (object\_order \wedge object\_send) \Longrightarrow ask\_payment)$

```
<policy>
    <actions>...</actions>
    <behaviour>
        <implication>
            <premise>
                <and>
```

```
                    <event>obj_order</event>
                    <event>object_send</event>
                </and>
            </premise>
            <conclusion>
                    <event>ask_payment</event>
            </conclusion>
        </implication>
    </behaviour>
</policy>
```

3. $\psi_3 : object\_send \Longrightarrow [low\_risk \wedge F^{-1}(negative)]$

```
<policy>
    <actions>...</actions>
    <behaviour>
        <implication>
            <premise>
                   <event>object_send</event>
            </premise>
            <conclusion>
                  <and>
                    <event>low_risk</event>
                    <sometime>
                        <event>negative</event>
                    </sometime>
                  </and>
            </conclusion>
        </implication>
    </behaviour>
</policy>
```

4. $\psi_4 : [high\_risk \wedge F^{-1}(object\_send \wedge pay\_cancel)]$
$\Longrightarrow pay\_TTP\_notified$

```
<policy>
    <actions>...</actions>
    <behaviour>
        <implication>
            <premise>
                <and>
                   <event>high_risk</event>
                   <sometime>
                        <event>object_send</event>
                   </sometime>
                   <event>pay_cancel</event>
                </and>
            </premise>
            <conclusion>
                    <event>pay_TTP_notified</event>
            </conclusion>
        </implication>
```

```
    </behaviour>
</policy>
```

Annexe B

Protocole et Négociation

La description donnée dans cette annexe ne présente qu'un extrait des programmes Java réalisés pour tester les exemples de cette thèse.

B.1 Protocole & Interface

B.1.1 Programme Serveur

```
import java.net.* ;
import java.io.* ;

public class server {

    public static void main(String [] args) {

ServerSocket srv ;
int port = 5555 ;
try {

         srv = new ServerSocket(port) ;
         Socket s = srv.accept() ;

         OutputStream os =  s.getOutputStream();
         InputStream is = s.getInputStream();

            ObjectInputStream ois = new ObjectInputStream(is);
         ObjectOutputStream oos = new ObjectOutputStream(os);

         Message msg_in = null;
         Message msg_out = new Message();

            // agent de négociation
            NegotiationAgent agent = new NegotiationAgent();

            // event analyzer
            EventAnalyzer analyzer = new EventAnalyzer();
```

```
            String event;

            while(true){
             try {
               msg_in = (Message) (ois.readObject());
                      if (msg_in.checkend()) break;

                      event=analyzer.MessageToEvent(msg_in);
                      agent.process(event);
                      event = agent.getEventCandidate();

                      msg_out = analyzer.EventToMessage(event);
                      oos.writeObject(msg_out)

             } catch ( Exception e){}
       }

        os.close() ;
    is.close() ;
    s.close() ;
} catch(IOException e) { }
    }
}
```

B.1.2 Programme Client

```
import java.net.* ;
import java.io.* ;

public class client {

    public static void main(String [] args) {
Socket s ;

// References de la socket
String hote = "localhost" ;
int port = 5555;

try {
    s = new Socket (hote, port) ;

    OutputStream os = s.getOutputStream();
    InputStream is = s.getInputStream();

        // le client envoie d'abord le message de demande
    ObjectOutputStream oos = new ObjectOutputStream(os);

        //Message de la première demande
    Message msg_out = new Message();
```

```
        String event;

    oos.writeObject(msg_out);

        // agent de négociation
        NegotiationAgent agent = new NegotiationAgent();

        // event analyzer
        EventAnalyzer analyzer = new EventAnalyzer();

        while (true){
           //ObjectInputStream pour recevoir le message du serveur
             ObjectInputStream ois = new ObjectInputStream(is);
             Message msg_in = null;

                 msg_in = (Message) (ois.readObject());
                 if (msg_in.checkend()) break;

                 event=analyzer.MessageToEvent(msg_in);
                 agent.process(event)
                 event = agent.getEventCandidate();

                 msg_out = analyzer.EventToMessage(event);
                 oos.writeObject(msg_out)

        }
    ois.close();
    oos.close();
    os.close();
    is.close();
    s.close();
}
catch (MessageException me ) {
    me.printMsg();
}
catch ( UnknownHostException e ) {
    System.out.println("Unknown host" + e);
}
catch ( Exception e) {
    System.out.println("IO exception" + e);
}
  }
}
```

B.1.3 Messages du protocole

```
    import java.io.*;
    import java.lang.Exception;
```

```
class MessageException extends Exception
{
    String msg;
    MessageException( String erreur ) {msg = erreur; }
    public void printMsg() {
        System.out.println("Erreur: " + msg);
    }
}

public class Message implements Serializable
{

    int ident; //identifiant du msg
    int type;  //type du message
    String event //événement équivalent
    String data; //données échangées

    public Message( int id, int t, String e, String m)
    throws MessageException{
     this.setIdent( id );
     this.setType( t );
     this.setEvent( e );
        this.setData(d);
   }

    public int getIdent() { return ident; }
    public int getType() { return type; }
    public String getEvent() { return event; }
    public String getData() { return data; }

    public void setIdent( int id ) throws MessageException{}
    public void setType( int t ) throws MessageException{}
    public void setEvent( String e) throws MessageException{}
    public void setData( String d ) throws MessageException{}

    //vérifier si c'est le message de FIN de l'échange
    // c'est le messagge de type Conclusion
    public boolean checkend() {return ;}

};
```

B.1.4 Event Analyzer

```
public class EventAnalyzer{
{
    String eventname[]; // tableau des événements
    Message msg_list[]; // liste de messages

    Message msg_in; //message d'entrée
```

```
    Message msg_out;  //message de résultat
    String event_in //événement correspondant avec le message d'entrée
    String event_out; //événement correspondant avec le message de résultat

    public EventAnayze() {};
    public Message EventToMessage(String event){return msg;}
    public String   MessageToEvent(Message msg){return event;}

    public Message getMessageIn() { return msg_in; }
    public Message getMessageOut() { return msg_out; }
    public String getEventIn() { return event_in; }
    public String getEventOut() { return event_out; }

    public void setMessageIn( Message msg )
                throws EventAnalyzerException{}
    public void setMessageOut(Message msg )
                throws EventAnalyzerException{}
    public void setEventIn( String in)
                throws EventAnalyzerException{}
    public void setEventOut( String out)
                throws EventAnalyzerException{}
}
```

B.2 Agent de négociation

B.2.1 Processus de négociation

```
import java.io.*;
import java.lang.Exception;

public class NegotiationAgent {

    public String event_in;
    public String event_candidate;
    public NegotiationStep sn;

    public NegotiationAgent() throws NegotiationAgentException{}

    //traitement d'une négociation avec un message d'entrée
    public void process(event) {
            sn = new NegotiationStep(event_in);
            event_candidate = sn.getCandidate();
      }

    public String getEventIn(){ return event_in }
    public String getEventCandidate(){ return event_candidate }
}
```

B.2.2 Étape de négociation

```
    public class NegotiationStep {
        public String event_in;
        public String event_candidate;
        public String candidate_set[];

        public NeogitiationStep(String event_in) {

            // Calculer l'ensemble des événements de candidates dans ES
            // en applelant la procédure de vérification de politique
            candidate_set = getListCandidate(event_in) // en cours

            // le candidat par la stratégie
            event_candidate = Strategy(event_in, candidate_set)
        }


        //retourner l'événement de candidat
         public getCandidate() {
            return event_candidate;
         }

        // Appliquer la stratégie
        public String Strategy(String event[], event_in) {
            String candidate;
            // Calculer le poids de causalité des événements de candidat
            // Prendre un événement dont le poids est plus petit
             return candidate;
        }
}
```

Glossaire

LTL : Linear Temporal Logic

PP-LTL : Pure Past Linear Temporal Logic

API : Application Policy Interface (KeyNote)

ABAC : Attributs Based Acces Control (RT framework)

RT : Role Based Trust Management Framework

SD3 : Secure Dynamically Distributed Dialog

JavaHBAC : Java History Based Security Manager

SECURE : Secure Environments for Collaboration among Ubiquitous Roaming Entities

DSD : Document Structure Description

XML : Extensible Markup Language

SSL : Secure Sockets Layer

SULTAN : Simple Universal Logic-oriented Trust Analysis Notation

PDF : Probabilistic Density Function

PAL : Property based Authentication Language

RAL : Role based Authorization Language

IP : Internet Protocole

IPSEC : Internet Protocol Security

TPL : Trust Policy Language

TNL : Trust Negotiation Language

DCM : Dynamic Checking Model

Bibliographie

[1] Introduction to risk analysis. 2001. http://www.security-risk-analysis.com/introduction.htm.

[2] Abdul-Rahman and S. Hailes. Supporting trust in virtual communities. In *In Proceedings of the 33rd Hawaii International Conference on Systemes Sciences*. IEEE Computer Society, 2000.

[3] Claude Auge and Pierre Larousse. Dictionnaire larousse. Larrouse, http ://www.editions-larousse.fr, 2000.

[4] Weston Union Bank. Transfers bancaires internationaux france. http://www.westerunion.fr, 2000.

[5] E. Bertino, E. Ferrari, and A. Squicciarini. Trust negotiation : Concepts, systems, and languages. *Computing in science & engineering 6 (4)*, pages 27–34, Jul-Aug 2004.

[6] E. Bertino, E. Ferrari, and A. Squicciarini. Trust-x : A peer-to-peer framework for trust establishment. *IEEE Trans. on Knowl. and Data Eng.*, 16(7) :827–842, 2004.

[7] Nancy Betencourt. Satisfaction des politiques de confiance. Ecole Nationale Superieure des Mines de Saint Etienne, Jun, 2010.

[8] Matt Blaze. Using the KeyNote Trust Management System, 03 2001. http://www.crypto.com/trustmgt/kn.html.

[9] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote Trust-Management System Version 2. Technical report, U. of Pennsylvania, 09 1999. http://www.cis.upenn.edu/~keynote.

[10] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote Trust-Management System, 2001. http://www.cis.upenn.edu/~keynote/index.html.

[11] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. KeyNote : Trust management for public-key infrastructures (position paper). In *Security Protocols Workshop*, volume 1550 of *Lecture Notes in Computer Science*, pages 59–63. Springer, 1998. http://www.cis.upenn.edu/ keynote/Papers/keynote-position.ps.gz.

[12] Matt Blaze, Joan Feigenbaum, and Martin Strauss. Compliance checking in the policymaker trust management system. In *Financial Cryptography*, pages 254–274, 1998. http://www.crypto.com/papers/pmcomply.pdf.

[13] Piero Bonatti, Claudiu Duma, Daniel Olmedilla, and Nahid Shahmehri. An integration of reputation-based and policy-based trust management. In *Proceedings of the Semantic Web and Policy Workshop*, November 2005. joint with 4th International Semantic Web Conference Galway,Ireland.

[14] V. Cahill and E. Gray et al. Using trust for secure collaboration in uncertain environments. IEEE Pervasive Computing, 2003.

[15] V. Cahill and J.-M. Seigneur et al. Secure : Secure environments for collaboration among ubiquitous roaming entities, ist-2001-32486. http://secure.dsg.cs.tcd.ie, 2002.

[16] V. Cahill and J.-M. Seigneur et al. The secure project. http://secure.dsg.cs.tcd.ie, 2004.

[17] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, 2000.

[18] B. Kracher Corritore, C.L. and S. Wiedenbeck. An overview of trust : Working document. Technical report, 2001. http://cobacourses.creighton.edu/trust/articles/trustpaper2-9-01_final.rtf.

[19] R. Lara D Olmedilla, A. Polleres, and H. Lausen. Trust negotiation for semantic web services. In *1st International Workshop on Semantic Web Services and Web Process Composition*, San Diego, 2004.

[20] Nathan Dimmock, Jean Bacon, David Ingram, and Ken Moody. Risk models for trust-based access control (tbac).

[21] Niklas Een and Niklas Soresson. Solver minisat. In *http://minisat.se*, 2005.

[22] Elena Ferrari Elisa Bertino and Anna Cinzia Squicciarini. X-tnl : An xml-based language for trust negotiations. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '03)*, pages 81–84, Villa Olmo, Lake Como, Italy, June 2003. IEEE Computer Society 2003.

[23] Blaze et al. Managing trust in an information-labeling system. In *European Transactions on Telecommunications*, 1997.

[24] G. Edjlali et al. History-based access control for mobile code. In *In Proceedings from the 5th ACM Conference on Computer and Communications Security (CCS'98)*. ACM Press, 1998.

[25] L. Olson et al. Trustbuilder as an authorization service for web services. In *International Workshop on Security and Trust in Decentralized/Distributed Data Structures (STD3S) in conjuction with the 22nd International Conference on Data Engineering (ICDE'06)*, Atlanta, Georgia, April 2006.

[26] M. Winslett et al. Negotiating trust on the web. In *IEEE Internet Computing*, November 2002.

[27] T. Ryutov et al. Adaptive trust negotiation and access control. In *10th ACM Symposium on Access Control Models and Technologies*, Stockholm, Sweden, June 2005.

[28] T. Grandison et M. Sloman. Trust management tools for internet applications. In *First international conference on trust management. Crète*, May 2003.

[29] S. Kent (BBN Corp) et R. Atkinson. ecurity architecture for the internet protocol. In *http ://tools.ietf.org/html/rfc2401*, 1998.

[30] J. Feigenbaum and P. Lee. Trust management and proof-carrying code in secure mobile code applications : Position paper. In *DARPA Workshop on Foundations for Secure Mobile Code*, 1997.

[31] R. Fielding and et al. Hypertext transfer protocol – http/1.1. In *http ://tools.ietf.org/html/rfc2616*, 1999.

[32] Diego Gambetta. In *Trust : Making and breaking cooperative relations*. Basil Blackwell, 1988.

[33] Diego Gambetta. Can we trust ? In *Trust, Diego Gambetta*. Basil Blackwell, 1988.

[34] E Gerck. Toward real-world models of trust. Technical report, E Gerck and MCG, 1998. http://www.mcg.org.br/trustdef.htm.

[35] T. Grandison and M. Sloman. Trust management tools for internet applications. In *Proceedings of the 1st International Conference on Trust Management*, 2003. LNCS, Springer.

[36] Tyrone Grandison. Trust management for internet applications, phd thesis. *http://www.doc.ic.ac.uk/ mss/Papers/Grandison-phd.pdf*, 2003.

[37] Amazone Group. Online shopping for electronics, apparel, computers, books, dvds and more. http://www.amazone.com, 2000.

[38] Ebay Group. Site de vente aux enchères des objets neuf ou d'occasions. http://www.ebay.com, 1995.

[39] D. Harkins and D. Carrel. The internet key exchange (ike). In *http ://tools.ietf.org/html/rfc2409*, 1998.

[40] K. Havelund and G. Rocu. Synthesizing monitors for safety properties. In *Tools and Algorithms for the Construction and Analysis of Systems : 8th International Conference (TACAS'02)*. Lecture Notes in Computer Science, pringer-Verlag, 2002.

[41] Afred Horn. On sentences which are true of direct unions of algebras. pages 14–21. Journal of Symbolic Logic, Vol. 16, No. 1. (1951), 1951.

[42] Housley and et al. Internet x.509 public key infrastructure, certificate and crl profile. In *http ://www.ietf.org/rfc/rfc2459.txt*, 1999.

[43] IBM. Ibm trust establishment policy language. http://www.haifa.il.ibm.com/projects/software/e-Business/TrustManager/PolicyLanguage.html.

[44] IBM. Access control meets public key infrastructure, or : Assigning roles to strangers. In *IEEE Symposium on Security and Privacy*. IEEE, 2000.

[45] PayPal Inc. Paypal. http ://www.paypal.fr/fr.

[46] ISO. Guide iso73 : Management du risque - vocabulaire - lignes directrices pour l'utilisation dans les normes. ISO, 2002.

[47] P Jaillon, M Roelens, X Serpaggi, and H Vu. Towards an approach for trust negotiation. In *In Sandeep Kirshnamurthy and Pedro Isaias edts, Proceedings of IADIS International Conference - eCommerce'07*, Dec, 2007.

[48] T Jim. Sd3 : a trust management system with certified evaluation. in ieee symposium on security and privacy. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2001.

[49] A. Josang and T. Grandison. Conditional inference in subjective logic. In *6th International Conference on Information Fusion*, July, 2003.

[50] Audun Josang. The right type of trust for distributed systems. In *ACM New Security Paradigms Workshop*, http://www.idt.ntnu.no/ ajos/papers.html, 1996.

[51] Audun Josang. Artificial reasoning with subjective logic. In *2nd Australian Workshop on Commonsense Reasoning*, http://www.idt.ntnu.no/ ajos/papers.html, 1997.

[52] Audun Josang. A subjective metric of authentication. In *5th European Symposium on Research in Computer Security (ESORICS'98)*, http://www.idt.ntnu.no/ ajos/papers.html, 1998.

[53] Audun Josang. An algebra for assessing trust in certification chains. In *Network and Distributed Systems Security Symposium (NDSS)*, San Diego, California : The Internet Society, 1999.

[54] Audun Josang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. In *Decision Support Systems*, 2006.

[55] Audun Josang and S.J Knapskog. A metric for trusted systems. In *Proceding of the 21st National Information Security Conference (NIST-NSCS 1998)*, 1998.

[56] N. Klarlund, A. Moller, and M. I. Schwartzbach. The dsd schema language. In *Automated Software Engineering*, August, 2002.

[57] Konstantin Korovin. Course on logic in computer sciences. In *www.cs.man.ac.uk/ korovink/cs2142/chapter14.pdf*. School of Computer Science, The University of Manchester., 2006.

[58] Karl Krukow. Project : Java security manager history based access control for untrusted java applications. 2006.

[59] Karl Krukow, Mogens Nielsen, and Vladimiro Sassone. A framework for concrete reputation-systems with applications to history-based access control. In *CCS '05 : Proceedings of the 12th ACM conference on Computer and communications security*, pages 260–269, New York, NY, USA, 2005. ACM.

[60] P Lamsal. Understanding trust and security. Technical report, 2001. http://citeseer.nj.nec.com/lamsal01understanding.html.

[61] Ninghui Li and John C. Mitchell. Rt : A role-based trust-management framework. In *Proceedings of The Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, pages 201–212, Washington, D.C, April 2003. IEEE Computer Society Press.

[62] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.

[63] Niklas Luhmann. Confiance et familiarite. problèmes et alternatives. In *Réseaux 2001/4 No 108*, pages 15–35. Lavoisier, 2001.

[64] J. Feigenbaum et J. Lacy M. Blaze. Decentralized trust management. Technical report, AT&T Research Labs, 1996. http://www.researchatt.com/resources/trs/.

[65] J. Feigenbounm M. Blaze and A.D.Keromytis. Keynote : Trust management for pulic key infrastructures. In *Cambridge 1998 Security Protocols International Workshop*, 1998.

[66] David Maier and et al. A database programming language. In *Workshop on logic and databases*, 1977.

[67] S. Marsh. Formalising trust as a computational cencept. In *PhD Thesis*, University of Stirling, 1994.

[68] D.H McKnight and N.L Chervany. The meaning of trust, technical report misrc. working paper series 96-04. Technical report, University of Minnesota, Management Information Systems Reseach Center, 1996. URL :http://misrc.umn.edu/wpaper.

[69] A. Moller. The dsd2.0 project website. In *http ://www.brics.dk/DSD*.

[70] E. lupu N. Damianou, N. Dulay and M. Sloman. The ponder policy specification language. In *Proc. of POLICY 2001*, volume LNCS 1995. Springer-Verlag, 2001.

[71] Mogens Nielsen and Karl Krukow. On the formal modelling of trust in reputation-based system. In *Theory Is Forever : Essays Dedicated to Arto Solomaa on the Occation of his 70th Birthday*, pages 192–204. Springer Verlag, 2004.

[72] D. Olmedilla P.A. Bonnatti. Policy language specification. In *REWERSE Deliverable I2-D2*, 2005. http://rewerse.bet/deliverable.html.

[73] Daniel Olmedilla Piero Bonatti. Driving and monitoring provisional trust negotiation with metapolicies. In *Proc 6th IEEE International Workshop on Policies for Distributed System and Networks(POLICY '05)*, 2005.

[74] Amir Pnueli. The temporal logic of programs. In *In Proceedings from the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*. IEEE, 1977.

[75] P.Zimmerman. Pgp user's guide. In *MIT Presse*, 1994.

[76] P.Zimmerman. Pgp documentations. In *Network Associates International*, 1999.

[77] W. Nejdl R. Gavriloaie, D. Olmedilla, K. Seamons, and M. Winslett. No registration needed : How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *1st First European Semantic Web Symposium*, Kerakson, Greece, May, 2004.

[78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2) :120–126, 1978.

[79] Vitaly Shmatikov. Reputation-based trust management. *Journal of Computer Security, special issue on selected papers of WITS 2003 (ed. Roberto Gorrieri), vol. 13(1)*, pages 167–190, 2005.

[80] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logic. In *Journal of the ACM*. ACM, 1985.

[81] W3C. Xml signature syntax and processing (second edition). http ://www.w3.org/TR/xmldsig-core.

[82] S. Week. Understanding trust management system. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'01)*, 2001.

[83] Vicki E. Jones William H. Winsborough, Kent E. Seamons. Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition DISCEX2000*, volume 1, pages 88–102, January 2000.

[84] M. Winslett. An introduction to automated trust establishment. In *1st International Conference on Trust Management*, Crete, Greece, May 2003.

[85] D. Omedilla W.Nejdl and M. Winslett. Trust : automated trust negotiation for peers on the web semantics. In *Workshop on Secure Data Management in a Connected World(SDM'04)*, 2004.

École Nationale Supérieure des Mines
de Saint-Étienne

NNT : 2010 EMSE 0585

Hoan VU

Trust Management Infrastructure for Internet

Specialty : Computer sciences.

Keywords : Security, security policies, trust, risk, trust management, risk management, trust negociation, e-commerce, security protocol.

Abstract :

Trust establishment is an important problem which often arises everyday. We need to assess the trust in someone or something before making decisions on their actions. It is also a very important problem for Internet applications where participants of a system are virtual entities. The trust establishment is a key factor for e-commerce applications and services which involve interactions with unknown users.

The objective of this thesis is to build an infrastructure for trust management which allows each participant to express his own security policy. The security policy is a way for the participant to define his own access control to his own resources and services. The infrastructure provides a trust negotiation mechanism that allows two participants to establish a mutual trust between them for interactions.

The important point of our proposal of an infratructure for trust management is that we use all available information such as credentials (signed certificates), reputations, recommendations or risk information about the peer to make decisions on trust. All these factors are expressed in the security policy by using our proposed policy language.

École Nationale Supérieure des Mines
de Saint-Étienne

NNT : 2010 EMSE 0585

Hoan VU

Infrastructure de gestion de la confiance sur Internet

Spécialité : Informatique

Mots clef : Sécurité, politique de sécurité, confiance, risque, gestion de confiance, gestion des risques, négociation de la confiance, e-commerce, protocoles de sécurité.

Résumé :

L'établissement de la confiance est un problème qui se pose en permanence dans la vie quotidienne. Nous avons toujours besoin d'évaluer la confiance que l'on a en quelqu'un avant de décider d'entreprendre une action avec. Il s'agit bien évidemment d'une question très importante pour les applications de l'Internet où il est de plus en plus rare d'engager une transaction avec des personnes ou des entités que l'on connaîtrait au préalable. La confiance est un élément clé pour le développement et le bon fonctionnement des applications de e-commerce et par extension de tous les services qui amènent à des interactions avec des *inconnus*.

Le but de cette thèse est de proposer une infrastructure de gestion de la confiance qui permette à chaque participant d'exprimer sa propre politique de confiance ; politique qui guidera le comportement des applications qui fournissent ou qui permettent d'accéder à des services. Cette infrastructure met en œuvre des mécanismes de négociation qui vont permettre d'établir une confiance mutuelle entre les différents participants d'une transaction.

Un des points importants de notre proposition est d'offrir un langage d'expression des politiques qui permet d'utiliser toutes les sources d'informations disponibles telles que les qualifications (*credentials*), la notion de réputation, de recommandation, de risque pour exprimer sa politique de confiance.